Data Reduction in Data Warehouses

by

Themistoklis Palpanas

Department of Computer Science
University of Toronto
Technical Report CSRG-476

Data Reduction in Data Warehouses

Themistoklis Palpanas

Doctor of Philosophy

Department of Computer Science

University of Toronto 2003

Abstract

Much research has been devoted to the efficient computation of relational aggregations. In this thesis we consider the inverse problem, that of deriving (approximately) the original data from the aggregates.

We motivate this problem in the context of two specific application areas, approximate query answering and data analysis. We propose a framework based on the notion of information entropy that enables us to estimate the original values in a data set, given only aggregated information about it. We then show how approximate queries on the data from which the aggregates were derived can be performed using our framework. We also describe an alternate use of the proposed framework that enables us to identify values that deviate from the underlying data distribution, suitable for data mining purposes.

We present a detailed performance study of the algorithms using both real and synthetic data, highlighting the benefits of our approach as well as the efficiency of the proposed solutions.

Subsequently, we consider the above problem in a space constrained environment, where only a subset of the required aggregates can be stored. More specifically, we wish to select a set of aggregates of interest, subject to a constraint on the total space occupied by these aggregates. The objective is to maximize the total benefit, which is a function of the number and importance of the queries that we can estimate given the selected aggregates.

For this problem, which as we show is NP-hard, there are no known polynomial approx-

imation schemes, and we propose several algorithms for solving it. We explore the use of greedy and randomized techniques as well as clustering based approaches. The solutions presented herein are generic and can be applied to other problem domains as well.

We explore the properties and special characteristics of the above techniques with an experimental evaluation. Our results illustrate the behavior of the algorithms under different settings, and highlight the benefits of each approach. Based on our analysis, we present worst case scenarios for the algorithms. This offers insight into the operation of the algorithms, and provides a practical guide for selecting among the proposed techniques.

Contents

List of Tables

List of Figures

Chapter 1

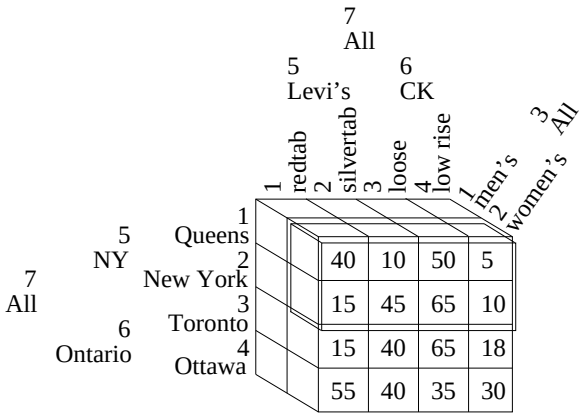
Introduction

In the following sections we discuss the motivation for our work, and introduce the specific problems that we are trying to solve. Then, we present a summary of the contributions we make in this thesis. Finally, we describe the outline for the rest of this document.

1.1 Approximate Querying and Exploration in Datacubes

In recent years, there has been an increasing interest in warehousing technology and OLAP applications which view data as having multiple dimensions, with hierarchies defined on each dimension. Users typically employ OLAP applications for decision making. They inquire about the values, and analyze the behavior of measure attributes in terms of their dimensions. Consider for example Figure 1.1(a) showing a typical simplified OLAP cube, with three dimension attributes (location, jeans, and gender), and hierarchies defined per dimension. Users, for analysis purposes, commonly inquire about the values of aggregates, like the total volume of sales of jeans in NY state. Such aggregates can be computed using the datacube operator [GBLP96], and queries of this kind can be efficiently supported.

The volume of data stored in OLAP tables is typically huge, on the order of many gigabytes, or even terabytes. In some cases, users store on disk only a subset of the data they own. They move the rest of the detailed data to tertiary storage systems, or even take them off line, while keeping only a small amount of aggregated data that are of interest. A common example is historical sales data, where only the data of the most recent years are stored online, and the rest are archived. When older years are archived, possibly data of finer granularity, such as sales of silvertab Levi's in New York city, are permanently



(a) The entire dataset.

		All			
		Levi's		CK	
		redtab	silvertab	loose	low rise
		55	55	115	15
NY	Queens 105	x	x	x	x
	New York 135	x	x	x	x

(b) Aggregated values for the selected portion of the dataset. That is, the sales of women's jeans in New York.

Figure 1.1: Example of a 3-dimensional data cube with sales data.

deleted, and only aggregated values are stored for future reference, for example, total sales of Levi's in each state. In other cases, even if the data remain online, they are aggregated not only to support user queries faster, but also to save space. For example, in several organizations, such as AT&T, terabytes of data are generated every day. These data are typically aggregated to save storage space, thus the detailed information that produced the aggregates is lost.

Given the summarized form of data, users are often interested in inquiring about the data from which the summarized form was generated. These data either might have expired from the warehouse, and thus are no longer available, or they might not be available online. In such cases, generating good estimates for the original data in response to queries is a pressing concern. Figure 1.1(a) depicts a 3-dimensional data cube, storing the total sales

of different kinds of jeans for men and women, across different cities. In this example, assume that for the women's jeans and for the state of NY (i.e., the upper front portion of the data cube), we only store the *aggregated* sales for each city and for each kind of jean as shown in Figure 1.1(b), and that we have deleted all the detailed values. The users might want to inquire about the number of redtab Levi's jeans sold in Queens NY (a point query), or they might request the number of women's jeans sold in each city of NY state (a range query). In the latter case, the answer consists of *all the individual* values marked as "x" in Figure 1.1(b). We want to be able to answer these queries approximately using only the stored aggregate values. In this work, we present a technique that addresses this problem. Similar issues arise in transaction recording systems [JMS95] as well as in statistical databases [AM92, Mal93].

Even if the original base data exist, the ability to reconstruct the original data from the summaries is of great value. Computing summaries is essentially a data reduction process in which great information loss takes place. In order to reconstruct the data, various assumptions have to be made about the statistical properties of the reduced data. Given the reconstructed and the original data at hand, we can test how valid our assumptions about the original data were, just by comparing the two. This is useful in reasoning about the properties of the underlying data set and could be of great value in data mining. It can help detect correlations in the data, and identify deviations, that is, values that do not conform to the underlying model. Such results are of great interest to the analyst, because they indicate local or global abnormalities.

In this thesis, we propose the use of an information theoretic principle for the reconstruction of the original data from the summarized forms. Since data in a warehouse are aggregated as a means of summarization, we examine the problem of reconstructing the original data from the aggregates.

Our reconstruction technique is based on the well recognized and widely applicable information theoretic principle of *maximum entropy* [KK92]. We present algorithms for the efficient reconstruction of data from the aggregates. Moreover, using an information theoretic formalism, we identify and describe an alternate benefit of the proposed reconstruction techniques, namely the ability to 'rank' each reconstructed value by its potential 'interest' to the user, as a means of aiding data analysis. The notion of interest in data mining is not well formulated and several approaches have been proposed for its formal definition

[ST96a]. Although all the measures of interest are subjective and heavily dependent on the application, we argue that an information theoretic approach to this problem, besides being mathematically rigorous, appears conceptually appealing as well.

1.2 Space Constrained Approximate Querying in Datacubes

It is often the case that not all data can be stored, because of their sheer size. In many cases even the size of the aggregated data is too large, so that portions of it have to be moved to tertiary storage, or permanently deleted. Then we have to choose which parts of the data to keep available.

In the discussion of our reconstruction technique in the previous section, we have assumed that we have enough space to materialize all the aggregates needed to reconstruct each query in our workload. In the case where only a subset of the aggregates can be materialized due to space constraints, one would be interested in materializing those aggregates that yield the most benefit in reconstructing queries while satisfying the imposed constraints.

The above situation leads to a space constrained optimization problem. We wish to fill the given space with the data that will give us the highest benefit. The benefit is determined by the number and importance of the queries that we can answer based on the stored data. The *view selection* problem [HRU96] in the context of data warehouses is an instance of such an optimization problem, and has been studied extensively in the literature.

In our case, where we want to select the aggregates that help reconstruct the most important queries, the optimization problem becomes more complicated. As we will discuss in more detail in Chapter 4, the benefit we get for including items in the solution is associated to *sets* of items, and not to individual items. In both the *Knapsack* and the *view selection* problems, the benefit increases with each item that is added to the solution. However, this is not true for the class of applications that we consider. In this setting there are additional constraints inherent in the problem, which dictate that when we select a new item to insert in the solution the added benefit is zero, unless a set of related items are inserted as well. These constraints make the problem harder, and in Chapter 4, we discuss how they affect the solution procedure. For the rest of the thesis, we refer to this optimization problem as the *Constrained Set Selection* problem.

Since there are no known polynomial approximation algorithms for this problem, we examine the use of known heuristic techniques, and propose new algorithms based on the greedy principle, randomization, and clustering (see Chapter 4). We experimentally study the performance and behavior of the proposed solutions under various conditions. The experimental evaluation is coupled with a theoretical analysis that serves as a practical guide for selecting among the techniques proposed herein.

1.3 Contributions

The contributions of this thesis, at a high level, are as follows.

- We propose a method for reconstructing multidimensional values based *only* on the aggregate data.
- We present a method to identify and rank deviations in multidimensional datasets that does not depend on any domain or user-specified parameters.
- We explore the properties and special characteristics of the above methods with an experimental evaluation, using both synthetic and real datasets.
- We examine the operation of the reconstruction technique under a space constraint, formulate associated optimization problems, and propose several algorithms for their solution.
- We evaluate the behavior of the above techniques with an experimental evaluation. We also provide an analysis of the algorithms that offers insight into their operation.

1.4 Outline

The outline of the thesis is as follows. Chapter 2 introduces the context for this work. In Chapter 3, we present the reconstruction algorithm. We discuss two particular application scenarios of the algorithm, and present experimental results evaluating the performance and the utility of the proposed techniques. In Chapter 4, we formulate optimization problems, and propose several algorithms for their solution. We evaluate their behavior using experimentation and theoretical analysis. In Chapter 5, we demonstrate the applicability of the

techniques discussed in this thesis in a single case study. Finally, we conclude in Chapter 6, and propose future research directions.

Chapter 2

Literature Review

During the past few years we have experienced a considerable growth in the amount of data produced and managed by different organizations worldwide. This growth has led in many cases to the introduction of multiple database systems within the same organization in order to deal with different aspects of the data [CCS93]. Nevertheless, the poor data analysis functionality that traditional database systems offer was the incentive for the advent and development of *data warehouse* systems. These systems store consolidated, historical, and summarized data, and are designed to support complex, mostly ad hoc queries, which may involve large portions of the stored data.

Typically, data warehouses use multidimensional models in order to effectively represent the wealth of information they manage [CD97]. The data is organized in dimensions which describe in a natural way most of the attributes associated with the data of interest. The dimensions are in turn organized into hierarchies, with data aggregated at each level, which enhance the functionality of the system. The operations that the system supports include increasing and decreasing the level of aggregation along any number of dimension hierarchies, selection and projection across dimensions and hierarchy levels, and defining various orientations of the multidimensional view of the data. This kind of technology is referred to as *On Line Analytical Processing* (OLAP), and we will discuss it further in the next sections.

Numerous traditional data mining techniques have been discussed in the OLAP context [Han98]. In this chapter, we give some background on OLAP and data mining. We argue that mining algorithms have to integrate more tightly with the OLAP environment

in order to harness its full power. We review different ways of identifying and extracting knowledge from large collections of data. We examine the notion of *interestingness* of data, that is, ways of determining what is interesting to the user and what is not. Subsequently, we explore algorithms and techniques that operate on large datasets and find the interesting portions therein, and we discuss the relationship between data mining and data approximation techniques.

Note that the following discussion is not intended to be exhaustive, but rather indicative of the different approaches and the future research directions in the area of data mining in data warehouses. For a more elaborate discussion refer to a previous study [Pal00]. In this chapter, we intend to set the context within which our work fits. Studies that are more specifically related to the problems presented in the next chapters are discussed at the end of the corresponding chapters.

2.1 OLAP Technology

2.1.1 Overview

The standard logical model for representing the data in a data warehouse is the *data cube*. A data cube is defined as the union of a set of aggregates derived from data stored in a relational table. The data cube offers a multidimensional view of all the data, which are represented as a set of numeric measures (or facts), and organized in several dimensions of interest [CD97]. The numeric measures are the values of the data we are interested in, and the ones on which the analysis will be performed. The dimensions are attributes of the data. They provide the context for the measures, and are organized in hierarchies of multiple levels. The measures are aggregated at each hierarchy level for each dimension to offer a more general view of the base data. Figure 2.1 depicts an example of a data cube. In this example the measures can be quantities such as total sales, revenue, and inventory.

The physical design of the above data model is different for *Multidimensional*- and *Relational*-OLAP servers (MOLAP and ROLAP respectively). MOLAP servers directly support a multidimensional view of the data, achieved through a multidimensional storage engine. This approach results in more natural ways of data representation, but requires special care when storing data since data sets in high dimensions tend to be sparse. On the other hand, ROLAP servers take advantage of the existing relational database technology.

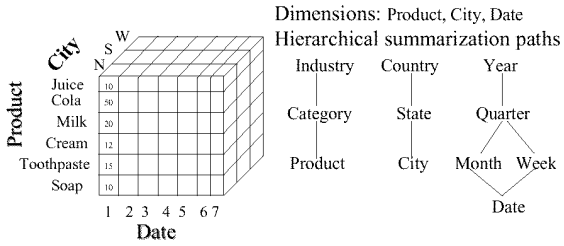
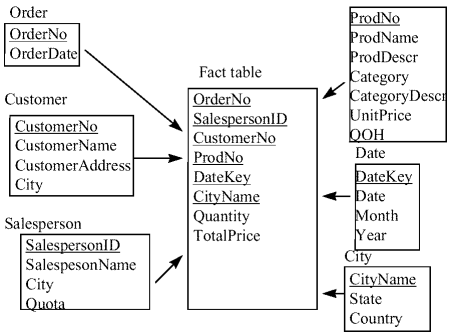
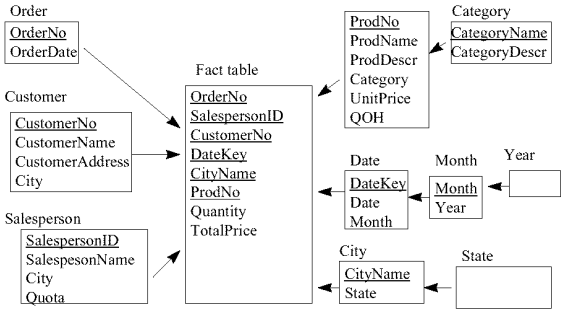


Figure 2.1: Example of a representation of a data cube.

The database consists of a fact table, storing all the measures, and dimensional tables around it. Having one table per dimension leads to a *star* schema, and by normalizing the dimension tables we get a *snowflake* schema (Figure 2.2).



(a) Star Schema



(b) Snowflake Schema

Figure 2.2: Two different approaches in the physical design of ROLAP servers.

2.1.2 The *CUBE* Operator

The problem of formalizing the computation of a data cube is discussed by Gray et al. [GBLP96]. This study focuses on relational database systems, and addresses the need

for an additional operator. The authors propose the *data cube* or *CUBE* operator, that computes the $0-, 1-, \ldots N-$ dimensional aggregates of a dataset with N aggregation attributes. This is a generalization of the SQL aggregation functions and the *GROUP BY* operator. The *CUBE* operator may be simulated by a union of 2^N *GROUP BY*s for N aggregation attributes. These *GROUP BY*s are depicted using the *lattice* notation [HRU96] (Figure 2.3), where each node (also called a *cuboid*) represents a *GROUP BY*. Nevertheless,

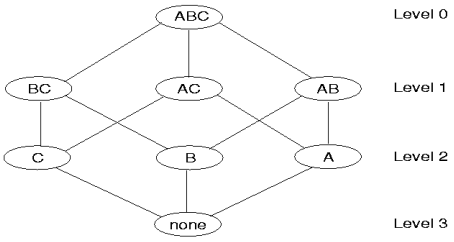


Figure 2.3: The lattice notation for a data cube with three dimensional attributes.

this approach is rather cumbersome and harder to write than using the *CUBE* operator. Furthermore, the final query would result in a large number of data scans and sorts, which is obviously inefficient. The *CUBE* operator allows the user to move between different levels of aggregation of the data by removing or adding aggregation attributes. Going up the levels is called a *roll-up*, while the opposite operation, going down the levels, is termed a *drill-down*. The aggregation functions can be classified into three categories [GBLP96]: *distributive*, which can be computed separately for disjoint subsets of the dataset and the partial results merged (e.g., count, sum, min, max); *algebraic*, which can be expressed using some of the distributive function (e.g., average); and *holistic*, which can only be computed for the whole dataset (e.g., median, rank).

Several algorithms have been proposed for the efficient computation of the data cube [AAD+96] [RS97] [ZDN97], and various indexing schemes for OLAP databases have been presented in the literature [SS94] [Sar97] [RKR97]. The problem of maintaining a data cube after we have computed it is also an interesting research question [MQM97] [KR99] [LSPC00] [PSCP02].

2.2 Data Mining

2.2.1 Overview

The framework that describes all the necessary steps involved in the process of knowledge extraction from databases is referred to as *Knowledge Discovery in Databases* (KDD). KDD is the process of identifying valid, novel, potentially useful, and ultimately understandable structures in data [Fay98]. In the above definition, data is the set of facts in the database, and structure refers to either a parsimonious description of a subset of the data, or a model representing the source that generated the data. The KDD process is comprised of many steps which involve data preparation, search for structures, knowledge evaluation, refinement, and consolidation with the existing expert knowledge. The term *data mining* refers to only one of the above steps. It is the step in the KDD process that, under acceptable computational efficiency limitations and selective evaluation criteria, enumerates structures over the data.

Mining association rules is one form of data mining that has attracted a lot of attention during recent years, and has motivated a great deal of work in the database community. It is essentially the search for dependencies which hold among items in a large dataset. The problem was first formulated by Agrawal et al. [AIS93] [AS94]. Several variations of the above algorithms try to minimize the required I/O operations [PCY95] [SON95] [Toi96] [BMUT97]. The association rule mining algorithms were also extended to take into account the hierarchies that are present on items [SA95], as well as the order in which the items appear in the dataset [AS95] [SA96].

Knowledge discovery involves solving the problem of data analysis by identifying interesting patterns in huge collections of data. Nevertheless, discovery systems generate a wealth of patterns, most of which are of no interest to the user [FPSM91]. Therefore, a measure that captures the amount of information that a discovered pattern conveys is essential, and in most cases this measure is defined by the user [ST96c] [KMR$^+$94] [PT98] [LHC97]. To this effect, some systems provide an interactive data mining framework, where the user is an active participant in the process [ST96b] [TUA$^+$98] [NLHP98]. Other studies present more automated ways of identifying the unexpected patterns in a dataset that require little or no human intervention [ST96c] [CSD98] [JM00].

For a more complete discussion of data mining techniques, the interested reader should

refer to surveys of the area [CHY96] [HK00].

2.2.2 Deviation Detection

An interesting category of unexpected patterns are the *deviants* or *outliers*. An outlier is "an observation that appears to deviate markedly from other members of the sample in which it occurs" [BL94]. This fact may raise suspicions that the specific observation was generated by a different mechanism than the rest of the data. This mechanism may be an erroneous procedure of data measurement and collection, or an inherent variability in the domain of the data under inspection. Nevertheless, in both cases such observations are interesting, and the analyst would like to know about them.

There is extensive literature in the statistics community regarding outlier detection [BL94]. Yet, this work is not directly applicable to databases since it pre-supposes that the user knows the distribution which generated the data. Based on this knowledge, statistical tests can examine the probability that the given distribution produced the data point under consideration, and accordingly accept it or reject it. The problem is that, in the context of databases, the distributions that produced the data are unknown, and extremely hard to estimate. Moreover, when dealing with large collections of data we need efficient algorithms that are fast and scale linearly or near-linearly with the dataset size. Thus, other methods for outlier detection are necessary. In the following paragraphs, we will discuss some of the techniques used in the database community.

Detecting Deviants In Databases

The problem of finding outliers in large, multidimensional datasets is studied by Knorr and Ng [KN98]. They introduce the notion of distance-based outliers $DB(p, D)$ which are defined as follows. An object O in a dataset T is a $DB(p, D)$-outlier if at least a fraction p of the objects in T lie further than distance D from O. The above definition is a generalization of the notion of outliers supported by statistical outlier tests for standard distributions. The advantage is that this approach does not require any prior knowledge of the underlying data distribution, but rather uses the intuitive explanation that an outlier is an observation that is sufficiently far from most other observations in the database. However, it does not provide a ranking for the outliers, which would be useful in some cases.

The study describes three algorithms that require the user to specify the parameters p

and D. Note that a suitable value for D can only be determined through experimentation. The first algorithm is a simple nested loop algorithm, whose complexity is linear in the number of dimensions d, but quadratic in the size of the dataset N. The second algorithm operates on in-memory datasets, providing complexity linear in N, but exponential in d. The third algorithm works with disk-resident datasets, and guarantees no more than three passes over the entire database. Nevertheless, the complexity still grows exponentially with the dimensionality, which renders this approach viable only on low dimensional datasets.

In subsequent work [KN99], the algorithms are extended to discover outliers in lower dimensionality subspaces of the data as well. The definition of outliers is augmented by the characterizations *strong* and *weak*. An outlier O in the attribute space $\mathcal{A}_\mathcal{P}$ is strong if there are no outliers in any subspace $B \subset \mathcal{A}_\mathcal{P}$. Otherwise O is termed a weak outlier.

Another algorithm for efficiently identifying outliers in large datasets is described by Ramaswamy et al. [RRS00]. In this study, outliers are the n data points whose distance from their k-th nearest neighbor D^k is the largest, across all data points in the dataset. The parameters k and n are user defined.

The algorithm works as follows. First, using a clustering algorithm, it partitions the dataset in clusters, and computes lower and upper bounds on D^k for the points in each cluster. Then, based on this information, only the clusters that can contain outliers are retained. In the final step, the outliers are computed from the points that belong in the remaining clusters. The use of a multi-dimensional index structure (R-tree in this case) allows efficient pruning of the search space, and makes the approach scalable.

One of the problems with the approaches mentioned above is that they fail to detect the outliers in cases where the data have different densities in different parts of the dataset. A method that can identify outliers based on the density of data points is presented by Breunig et al. [BKNS00]. This approach does not assign labels to data points, that is, "outlier" or "not outlier". It rather computes for each point the *degree* of being an outlier, which is called the *Local Outlier Factor (LOF)*. The computation of the LOF value for a point x is based on the the density ρ_x of points around x, and the density $\rho_{x'}$ of points around x', where x' is a point in the set of the k-nearest neighbors of x. Then, the smaller ρ_x is when compared to $\rho_{x'}$ (for all k-nearest neighbors), the larger the LOF value is for

point x. Outliers are termed the points with the highest LOF values.

Arning et al. [AAR96] describe a technique which identifies *exceptions* (set of tuples) among the tuples of a large database. In a database with a set of items I, an exception set I_j is the subset of items that contributes the most to the dissimilarity of I. The goal is to find the exception set I_j with the smallest cardinality.

The proposed algorithm uses a dissimilarity function $\mathcal{D}(I_x)$ that measures the degree of relevance of the items in I_x. It may be any function that returns a low value if the elements in I_x are similar to each other, and a high value otherwise. An example of such a function for numerical data is the variance. Obviously, the challenge is to devise a function that will effectively capture the notion of similarity, and at the same time be computationally efficient. Nevertheless, it is not obvious what the choice of function $\mathcal{D}(\cdot)$ should be in various domains, and which class of functions can capture local phenomena as opposed to global. This point is important, since in the latter case we will end up identifying merely the extreme points of the dataset.

2.3 Approximate Query Answering

2.3.1 Overview

The rapid growth in the size of databases outpaces the technological advances in disk access [GS00]. This results in degradation of performance. One of the ways to remedy this situation is *data reduction*. Data reduction techniques are analogous to lossy compression schemes. They try to summarize huge collections of data using only small amounts of storage space. The summaries are supposed to capture the intricacies of the dataset, and be able to provide estimates of the actual data. In approximate answering, we are willing to sacrifice some of the accuracy for the sake of faster response times and reduced storage requirements.

In this section, we examine the relationship between summarization and data mining. Then, we review some of the work relevant to approximate query answering. The focus will be on histogram techniques, which have attracted a lot of attention and are widely used. A more extensive discussion of the summarization and approximation methods available can be found elsewhere [BDF+97].

2.3.2 The Relationship to Data Mining

Before attempting to establish the connection between data mining and data reduction, it would be useful to answer another question first: is data summarization related to approximate query answering? The answer to this question is affirmative. Indeed, there are a number of summarization techniques that can also be used to produce estimates of the actual data [BDF+97]. However, we do not argue that every data reduction method can be used for approximate query answering. For the rest of this discussion, we will consider the methods that belong to the intersection.

The role of data mining is to discover general patterns that describe the data. (We will ignore the fact that sometimes we are only interested in a small subset of those patterns.) These patterns may have the form of rules, or some model. Each of the generated patterns represents a subset of the raw data. Therefore, this procedure of identifying representative patterns in the data may be viewed as a means to achieve data summarization [FPSM91] [Man99].

Apparently, it is not clear what falls in each category, and under which conditions some method may be characterized as a data mining, or data summarization method, or both. Nevertheless, we argue that in some circumstances we can use a technique to achieve both the goals of knowledge extraction and data reduction (or data estimation). An example that illustrates the above point is depicted in Figure 2.4. In this example, a two-dimensional dataset exhibits a correlation between the two attributes. We may use a data mining technique to identify this pattern and produce a corresponding model (Figure 2.4(b)). Then, the derived model can be regarded as a concise summary of all the data points in the dataset (Figure 2.4(c)), allowing us to save space by storing only the model of the data. Finally, the same model can be used for approximate query answering (Figure 2.4(d)), that is, to estimate the value of one attribute given the value of the other in the absence of the real data.

2.3.3 Approximation in Databases

Histograms approximate the data in one or more attributes of a relation by grouping attribute values into buckets, and approximating the real attribute values (or frequencies) in the data based on summary statistics maintained in each bucket. The mathematical formu-

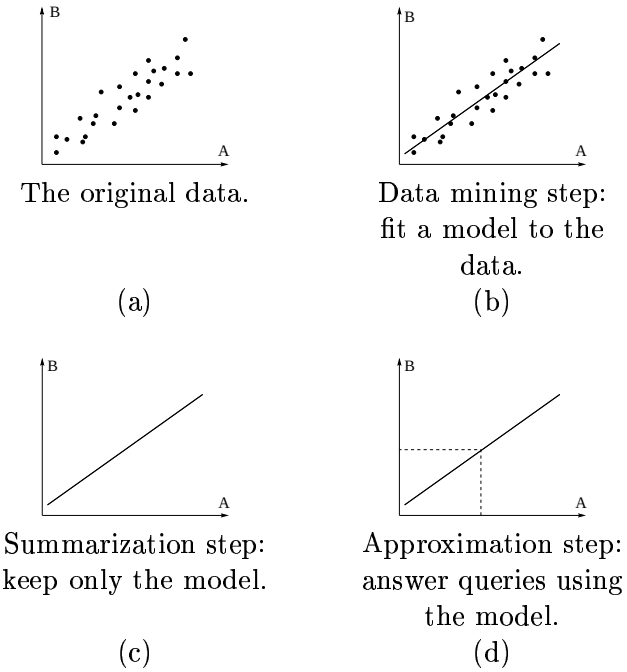


Figure 2.4: From data mining to approximate query answering.

lation of the problem is as follows. Let $\mathcal{D}$ be the (finite) domain of an attribute X in relation R, and the value set $\mathcal{V} \subseteq \mathcal{D}$ be the set of values of X actually present in R. We write $\mathcal{V}$ as $\mathcal{V} = \{v_i : 1 \leq i \leq M\}$, where $v_i < v_j$ for $i < j$. The set $\mathcal{V}$ can be represented as an M-long vector V. We will refer to any sequence of contiguous elements of V as a segment. We want to construct a summary vector H by selecting $B < M$ non-overlapping segments (buckets) that together include every element in V. We also provide a reconstruction function $R(\cdot)$ that approximates vector V (of length M) from vector H (of length $B < M$), and an error function $E(\cdot)$ that measures the distance between V and $R(H)$. The challenge is to find the vector H of length B which minimizes $E(H)$, given the initial vector V.

Several types of histograms have been suggested in the literature. Among those are the *equi-width* and *equi-depth* histograms. Equi-width histograms choose bucket boundaries in such a way that all bucket ranges are equal, and equi-depth ones guarantee that the total number of tuples mapped to each bucket is (nearly) the same for all buckets. More recently, new types of histograms proved to perform better [PIHS96b]. These are: *maxdiff*, which places the $B-1$ bucket boundaries between the v_i values with the $B-1$ maximum differences; *v-optimal*, that minimizes the variance of the values in each bucket; and *compressed*, which

dedicates an entire bucket to each one of the K highest values in V, and organizes the remaining $B - K$ buckets so that each of them has nearly the same total sum of values.

Note that all of the above approaches do not directly address the problem of minimizing $E(H)$, but rather provide approximations. Jagadish et al. [JKM+98] describe an algorithm that finds the optimal solution in time quadratic in M and linear in B. This is achieved using a dynamic programming algorithm that recursively finds for each segment the optimal split point (i.e., the point that minimizes the overall $E(H)$). The same algorithm can be augmented to provide quality guarantees on the maximum error of the estimations.

A subsequent study [JKM99] uses the above algorithm for deviation detection in time series. The main idea comes from the information theoretic principle of representation length. Deviants are termed the points whose removal from the sequence causes the largest reduction in the representation cost of the series.

The algorithm employs histograms for the representation of the time series V. Then, the problem is to find the points that once removed will decrease $E(H)$ the most. Note that, when we remove a data point from V, we decrease the number of buckets by one to account for the separate storage required for it. It turns out that the above dynamic programming algorithm can be adapted to solve this problem, albeit, in time quadratic in the size of the dataset. The advantage of this approach is that it can effectively capture deviant points in local contexts, that is, in relation to their neighbours, as well as from the global perspective. It is also interesting to note that the above technique (storing the deviant points in separate buckets) leads to more accurate histograms for the same storage space.

Matias et al. [MVW98] propose the use of a multiresolution wavelet decomposition technique to approximate a data distribution. The (Haar) wavelet transform takes as input a signal $X = \{x_1, \ldots, x_n\}$, and computes the averages and pairwise differences of all the pairs (x_{2i}, x_{2i+1}). Then, it recursively replaces each pair of points with their average value, and keeps the differences (along with the index of the point position) which are the *coefficients* of the transformed signal. In order to get the signal back, we only need the very last point value (where the recursion stopped), and the resulting coefficients. It turns out that we can disregard any coefficients with sufficiently small magnitude at the cost of incurring a small

error.

For a given storage space, the algorithm has to decide which m coefficients to keep for the representation of the original data. The study describes a set of greedy heuristics for this task. The complexity of the algorithm is linear in $|\mathcal{D}|$ (since the transformation requires padding all points in $\mathcal{D} - \mathcal{V}$ with zeroes). This is approximately the cost of answering queries as well (assuming $m << |\mathcal{D}|$). The experimental evaluation shows that this method performs better than *maxdiff* histograms in the general case. However, a more extensive comparison among the different approaches is necessary.

Faloutsos et al. [FJS97b] investigate a problem relevant to the histogram techniques. Histograms can be viewed as a summarized representation of the data. This study describes a method for estimating the original detailed data from the stored summary. Obviously, the problem is under-specified. One way of making up for the missing equations is to assume uniformity of values. However, the study shows that this assumption leads to poor estimates of the real values. Instead, the authors propose the technique of *linear regularization* which essentially tends to smooth the distributions it approximates. In other words, adjacent values in the solution will only have a small difference. Experiments show that this is a valid assumption for many datasets and results in better estimates. The algorithm has complexity linear in the size of the dataset. However, it does not scale up to many dimensions.

2.3.4 Approximation in Data Cubes

The problem of providing approximate answers for data cube structures has not been addressed sufficiently yet. The two methods outlined in the following paragraphs are merely extensions of previous approaches to many dimensions.

Poosala and Ganti [PG99] propose the use of the *MHIST* multidimensional histograms [PI97], which are the multidimensional version of *maxdiff*. The *MHIST* histogram starts by considering the whole data space as a single bucket. Then, it splits along the dimension whose 1-dimensional aggregate distribution contains the two adjacent points with maximum difference. The algorithm exploits the lattice structure of a data cube, and constructs histograms for only a subset of the cuboids (the rest are approximated using the computed histograms). The solution is based on a greedy heuristic, and runs in time exponential in

the number of attributes in the data cube.

The wavelet transform, which extends naturally to many dimensions, is explored by Vitter et al. [VWI98]. When considering multiple dimensions, the wavelet transform is applied on each dimension in a sequential manner (i.e., one dimension at a time). The complexity of the algorithm is quasi-linear in the total number of cells of the data cube, and there is an additional cost for answering queries, which depends on the cardinality of the dimensions involved in the query. This approach was also extended to accommodate sparse data cubes [VW99], and incremental updates to the model [MVW00].

The experimental results of the above two methods are not comparable, because different error measures are used in the studies. Once again, extensive studies should be performed in order to evaluate the different approaches.

A recent study [LPH02] focuses on the problem of finding succinct summaries of a data cube. The idea behind this approach is to leverage the regularities that exist among the values in the data cube, in order to compute a summarized form for it. The premise is that this summary preserves all the information in the original data cube, and that it reveals the semantics in the cube. That is, it exposes trends that hold over the data at different granularities, and along different dimensions.

The study describes algorithms that compute the summary of a data cube. This summary has the desirable properties that it maintains the cube semantics and lattice structure. The experimental evaluation of this approach, over synthetic and real datasets, shows that it can offer significant reductions in the space required for storing the data cube.

2.4 Data Mining in Data Cubes

2.4.1 Overview

The OLAP paradigm was introduced as an efficient and effective means to do data exploration [CCS93]. The premise was that the multidimensional view of the data along with the flexibility of the data cube operator would constitute a viable solution for the decision support problem. However, the sheer size of data cubes renders the task of intelligent analysis (i.e., anything more than just selecting subsets of the data cube, and applying the drill-down and roll-up operations) formidable. This fact merely transformed the problem

of having to study a wealth of information from bottom layer databases to decision support OLAP cubes. Hence, the need is now becoming apparent for intelligent analysis of data cubes as well.

Data mining is the research area that has the potential to address the above need. Numerous techniques have already been proposed for knowledge extraction, and many lessons have been learnt during this process. Nevertheless, there is still very little work done in the specific area of data mining in data cubes.

2.4.2 The General Case

The intersection of data mining and OLAP is extensively discussed by Han [Han97]. The author describes the *DBMiner* application which integrates OLAP technology with data mining techniques. Algorithms for performing characterization, classification, clustering, and association rule mining are embedded in the OLAP sub-system of *DBMiner*.

Such a system is useful [ZXH98], and a wide range of applications can benefit from it. Nevertheless, one would expect that all the preprocessing steps and the specialized structure of data cubes could be employed in a more integrated way. That is, we should be able to identify patterns in the data that could not possibly be discovered from the same data in raw format (i.e., not organized in a data cube). In the next section, we discuss an approach that follows this path.

2.4.3 Detecting Deviants In Data Cubes

Sarawagi et al. [SAM98] investigate the problem of identifying outliers in data cubes. The current practice in OLAP systems is to facilitate hypothesis-driven exploration of data cubes, where the analyst tries to find interesting information by simply using the data cube operators. This study proposes instead the *discovery-driven* exploration paradigm. In this environment, it is the system that recommends to the user potentially interesting paths of exploration in the data cube.

The algorithm mines the data for exceptions, and summarizes the results at different levels of the hierarchy. Exceptions are termed the cell values (of any aggregation level) that differ significantly from the anticipated value calculated using a model that takes into account all the aggregates in which the value participates. The model used in this study is similar to table analysis methods used in the statistics community.

Experiments with real datasets exhibit the flexibility of the algorithm in identifying outliers. Furthermore, the algorithm is transformed so that it can be incorporated in the computation of the datacube. This is crucial, since the complexity of the algorithm is prohibitive otherwise. Even then, the overhead introduced by the computation of the model is significant (up to more than 100%). Note that this approach requires that the whole datacube be computed, which is not a common practice, especially for large datasets. Moreover, the algorithm cannot update the model as new data arrive.

A subsequent study [Sar99] describes a method that produces an informed explanation for drops or increases that are observed in the data. More specifically, the *DIFF* operator is introduced, which explores the reasons for which a certain aggregated quantity is lower or higher in one cell compared to another. The reasons are expressed in the form of other cell values, belonging to aggregation levels of finer detail, that are most responsible for the difference under investigation. The challenge here is to not simply report all (or even the largest) changes in the detailed data, but rather provide a concise explanation. The former approach would either produce too many results or account for a small portion of the difference, while the latter has a high information content because it summarizes rows describing similar changes.

The problem is formulated in information theoretic terms as follows. A sender has a cube C_b and wants to transmit it to a receiver who already knows about cube C_a. In addition, the receiver has a summary $\mathcal{A}$ of N values, that describes the differences of the cubes. Then, the objective is to find the $\mathcal{A}$ (i.e., determine which N tuples to choose) which minimizes the amount of additional information that the receiver has to get in order to reconstruct C_b without errors. The solution is based on a dynamic programming algorithm.

Equally interesting to the *DIFF* operator is the *RELAX* operator [SS01], which can be thought of as the opposite of *DIFF*. The goal is to start from a problem case at a detailed level, and then have the system generalize to the broader context in which the problem occurs. The *RELAX* operator uses similar techniques as *DIFF*, but it offers a more flexible and general framework.

The problem of exploring large multidimensional datasets is studied by Sarawagi [Sar00]. This work proposes an automatic method for guiding a user analyst in the exploration of a

data cube. The system keeps track of the parts of the data cube that the user has visited, and suggests new exploration paths that will lead to the most surprising, unvisited, parts of the data.

The algorithm is based on the principle of Maximum Entropy, and suggests to the user to visit this part of the data for which the knowledge she already has is the most erroneous. In order to reduce the complexity of the algorithm, it considers only one new dimension at every step, which may lead to sub-optimal paths of exploration. Nevertheless, this technique offers an effective way of quickly navigating in the interesting parts of a massive dataset.

Chapter 3

Approximate Querying and Exploration of Data Warehouses

We now focus on data warehouses, and, more specifically, on data cubes. The volume of data stored in OLAP tables is typically huge, often on the order of multiple terabytes to petabytes. It is common practice to aggregate data in order to save storage space, a procedure during which the detailed information that produced the aggregates may be lost. Nevertheless, users are often interested in inquiring about the data from which the summarized form was generated. In this chapter, we present a reconstruction technique that produces estimates for the detailed values based solely on the aggregated information. We also evaluate the use of the same technique for identifying values of interest to a user analyst.

3.1 Contributions

The contributions we make in this chapter are as follows.

- We propose a method for reconstructing multidimensional values from aggregate data. In the OLAP environment our technique uses the already computed aggregated values.
- We describe an extension to the above method, in which we are able to provide quality guarantees (error bounds) for the reconstruction. Moreover, the quality of the reconstructed information can be controlled by the user, achieving any degree of desired accuracy at the cost of using more space.

- We present a method to identify and rank deviations in multidimensional datasets, that is, values that do not follow in general the underlying data distribution. These values are of particular interest to an analyst since they indicate local or global abnormalities. The power of the method we propose is that it does not depend on any a priori or domain knowledge for the problem at hand, and it also does not require any parameter settings or calibrations.

- The properties and special characteristics of the above methods are explored with an experimental evaluation, using both synthetic and real datasets.

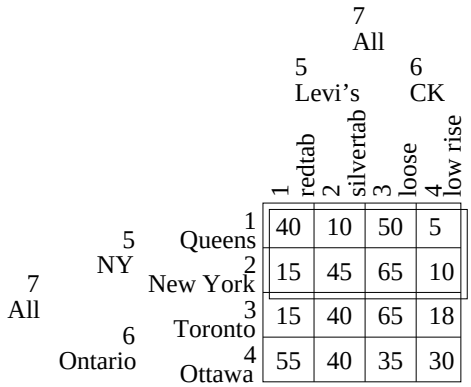
3.2 Outline

The outline of the rest of this chapter is as follows. In Section 3.3, we present some background material necessary for the rest of the chapter and Section 3.4 presents the reconstruction algorithm. Section 3.5 discusses the two particular application scenarios of the algorithm, and associated optimization problems. In Section 3.6, we present experimental results evaluating the performance and the utility of the proposed algorithms. Section 3.7 reviews related work, and finally we present our conclusions in Section 3.8.

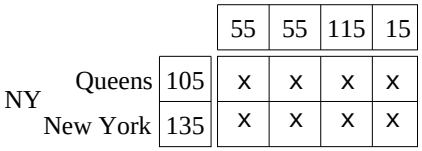
3.3 Background

Consider a relation schema $R = (A_1, A_2, \ldots, A_n, Y)$ and r an instance of R. $A_1, \ldots, A_n$ are dimension attributes and Y is a measure attribute. Attribute Y could represent volume of sales, dollar amount, number of calls, etc. Usually, each dimension of a datacube is associated with a set of hierarchically-related attribute values. A combination of attribute values from the leaves of the hierarchy specify a single data value in the dataset. Any combination of attribute values in which at least one comes from a non-leaf level of its hierarchy specifies a (hyper-rectangular) set of data values.

Following the above framework, attributes $A_i, 1 \leq i \leq n$ include dimension attributes and hierarchies possibly defined on them. For example, for the simplified OLAP table depicted in Figure 3.1(a) the schema is R(STATE, CITY, BRAND, PRODUCT, SALES). Values of attribute CITY are hierarchically grouped into values of STATE, and values of attribute PRODUCT are hierarchically grouped into values of BRAND.



(a) The entire dataset.



(b) Aggregated values for the upper half of the dataset.

Figure 3.1: Example dimension hierarchies on two dimensional sales data.

3.3.1 Grid Queries

With a schema of n attributes, assume that h attributes define hierarchies on the remaining $d = n - h$ attributes. We can enumerate the nodes of hierarchy $i, 1 \leq i \leq d$ starting from the leaves of the hierarchy and working towards the root node, such that $h_i = j$ denotes the jth node of hierarchy i. Viewing the table of Figure 3.1(a) with its dimensions and hierarchies as a multidimensional grid, following Jagadish et al. [JLS99], we refer to the d-dimensional vector $(h_1, \ldots, h_d)$ as a *grid query*. When a grid query specifies a single data value in the dataset, we refer to it as a *point* query. Otherwise, we call it a *range* query. For example, the grid query $(1, 2)$ is a point query asking for the sales of silvertab Levi's in Queens, while the grid query $(5, 7)$ is a range query referring to the entire upper half of the dataset. Grid queries are the most common queries in warehouse environments [JLS99].

For the rest of this thesis, we restrict our attention to range grid queries, and we will refer to them simply as queries. The reason we are interested in this special class of queries is that it significantly reduces the space of possible queries, which can be prohibitively large. The value of the above assumption will become apparent in Chapter 4, where the number of queries is part of the optimization problem.

It is important to note that whenever we refer to the term "query" in this chapter we refer to the *region of the dataset* that the query defines. For example, when we estimate or answer a query, it means that we provide approximate answers for each one of the data values that fall inside the region specified by the query. Then, by answering a range grid query using our techniques, we can also answer any other query asking for a subset of the answer we computed.

3.3.2 Dataset Values as Probability Distributions

A basic observation about any instance r of R is that it can be viewed as a discrete n dimensional probability distribution, $P_r(A_1, \ldots A_n)$. This can be accomplished by normalizing the value Y on each row of r by the sum of all Y values. The analogy between r and P_r can be extended further. We can derive from r all the distributions of P_r where we have eliminated any of the attributes participating in P_r by summing over them. We call this class of distributions *marginal distributions* or simply *marginals*. More formally, the marginal distribution of a set of random variables is the distribution obtained by summing the joint distribution over all values of the other variables.

Consider for example the following query:

SELECT $A_1, A_2, \ldots, A_{n-1}, sum(Y)$

FROM r

GROUPBY $A_1, A_2, \ldots, A_{n-1}$.

The outcome of this query is one of the n marginal distributions of P_r of order (number of attributes) $n-1$. All we need to do, is normalize $sum(Y)$ by the sum of all Y values.

Reasoning similarly, one can draw an analogy between all the group-by's on r and all the marginal distributions of P_r. Thus, we can view the problem of reconstructing r from its aggregates as analogous to the problem of reconstructing an n-dimensional probability distribution from a number of its marginal distributions. Based on this analogy, in the rest of this thesis, we use the terms group-by on instance r and marginal distribution of P_r interchangeably.

3.3.3 Maximum Entropy Distributions

For the following discussion, we will need the notion of *entropy*, which is defined with respect to a random variable Z as follows. If variable Z takes on the values $(z_1, z_2, \ldots, z_n)$ with

probabilities $(p_1, p_2, \ldots, p_n)$, then the entropy of variable Z, $H(Z)$, is

$$H(Z) = H(p_1, p_2, \ldots, p_n) = \sum_{i=1}^{n} p_i \log p_i.$$

Let $P(A_1, \ldots, A_n)$ be an n dimensional discrete probability distribution, to be estimated from a number of its marginals. With n variables, there are $2^n - 2$ marginals (excluding the grand total and the base data) of P in total. Moreover, there are $\binom{n}{k}$ marginals with k variables (equivalently of order k). Let S be an arbitrary subset of the powerset of $X = \{A_1, \ldots, A_n\}$. The problem of *maximum entropy estimation of P* is defined as follows:

Problem 3.1 [The Maximum Entropy Estimation Problem] *Find P such that it maximizes the entropy $H(P)$ of P, over all probability distributions that satisfy the following conditions:*

- *every element in P(X) has a non-negative value,*
- $\sum P(X) = 1$, *and*
- $\forall i \in S, \sum_{j \in \{S-i\}} P(j) = P(i)$,

where $P(j)$ is a marginal distribution of P.

The maximum entropy estimation of P is a model fitting technique. It has a unique solution [KK92], and it finds the model with the 'least' information or fewest assumptions given the specified constraints, which are the marginal distributions in our case. The method appears ideal for this estimation problem as it is designed for lack of data rather than an excess thereof. The overriding principle in maximum entropy is that when nothing is known the distribution should be as uniform as possible, and the participating attributes independent. The constraints specify the regions where the estimated distribution should be minimally non-uniform, as well as the attribute correlations that should exist in the estimated distribution.

In the definition of the maximum entropy estimation problem (Problem 3.1), the only constraints that serve as input to the problem are the given marginals, to which the solution should conform. In the general case, these constraints can take other forms as well. For example, instead of using just the sums of the detailed values (i.e., the marginals), we could also use any of the higher order moments, such as variance and skew. In this work, we restrict our attention to the use of marginals only, because these are readily available in a

data cube structure.

The only exception to the above restriction is in order to take into consideration the values of particular elements of the sample space, for which the exact *real* values are known. Then, there is no need to produce estimates for these elements. We discuss cases where the above situation is applicable in Sections 3.4.4 and 6.2.

3.3.4 Properties of Maximum Entropy

Let $P(X)$ be an n-dimensional probability distribution, and M_i be the set of all marginals of order i, $1 \leq i \leq n-1$ of $P(X)$. We denote by P_i, $1 \leq i \leq n-1$ the maximum entropy approximation to P using only the marginals in M_i as constraints. Then the following theorems hold.

Theorem 3.1 *[Kul68] Let C be the set of all n-dimensional probability distributions that have the same marginals as those in M_i, and assume that all distributions in C are equally probable to be the true distribution P. Then, the distribution $p = P_i \in C$ minimizes the expected distance among p and $P' \in C$, where the distance function is defined as:*

$$D(p, P') = \sum_X p(X) \log \frac{p(X)}{P'(X)}. \tag{3.1}$$

The measure D is known in the literature as the relative entropy [CT91] and measures the similarity of two probability distributions. More precisely D is a measure of the effort required to describe distribution P' based on the knowledge of distribution p. We can show that by minimizing $D(p, P')$ we also minimize the χ^2 test between p and P' [Kul68].

Theorem 3.2 *[Kul68] Let $P_i, 1 \leq i \leq n-1$ be the maximum entropy estimation of P using only marginals of order i. Then the following inequality holds:*

$$D(P_1, P) \geq D(P_2, P) \ldots \geq D(P_{n-1}, P). \tag{3.2}$$

Theorem 3.2 states that a better estimation of P can be performed by using marginals of order $i+1$ than marginals of order i, for $1 \leq i \leq n-1$.

3.3.5 Computing the Maximum Entropy Solution

Problem 3.1 is a constraint optimization problem that is not amenable to a general closed form solution. The problem is in general under-constrained, that is, there are more unknown

variables than equations. The standard technique for solving this maximization problem is the method of *Lagrange Multipliers* [Ber82]. This method requires the formation of equations based on the optimization objectives and the constraints imposed, and solve a rather complex system. In its one dimensional (single variable) version, this problem is amenable to an efficient solution [FJS97a]. However, the efficient solution of the one dimensional case does not generalize to multiple dimensions. Moreover, in order to solve this problem in the multidimensional case, a different system of equations has to be derived each time, depending on the specified constraints (marginal distributions available). This is not very appealing for automation. Ideally, we are after an algorithmic approach that can operate directly on the specified constraints and derive the estimation with no human intervention.

3.4 Algorithmic Solution

In this section, we propose the use of an algorithmic approach for the solution of the maximum entropy estimation problem (Problem 3.1). The technique is called *Iterative Proportional Fitting* (IPF), and was introduced by Deming and Stephan [DS40]. It is an iterative algorithm that converges to the maximum entropy solution. IPF has the following properties [BFH75].

1. It always converges monotonically to the required unique maximum entropy estimation, given a number of marginals.

2. A stopping rule may be used that ensures accuracy to any desired degree in the solution (i.e., in the asymptotic estimates).

3. The estimates depend only on the given set of marginals;

4. convergence, as well as the speed of convergence, are not directly affected by the starting values.

5. In some cases, convergence to the exact estimates is achieved after only a single iterative cycle[1].

[1] A complete discussion of this topic may be found elsewhere [BFH75].

6. It does not require the values of the marginals (or equivalently of the dataset) to be normalized in order to form a probability distribution.

3.4.1 Description of IPF

Let $r(A_1, A_2, \ldots, A_n, Y)$ be a relational instance. We specify a multidimensional region of interest $[d_{1_s}, d_{1_e}] \times \ldots \times [d_{d_s}, d_{d_e}]$, defined by a d-dimensional grid query $Q = (h_1, \ldots, h_d)$. Let S be the set containing all the marginals required for the reconstruction of Q, and k be the cardinality of S. To illustrate the above with an example, we use the dataset of Figure 3.1(a). The highlighted box in the figure defines a range query; the two marginals based on which we will estimate this query are the row and column aggregates shown in Figure 3.1(b).

Let $P_i(j), 1 \leq i \leq k, j \in S$, be one of the i marginals of P. Note that by j we refer to a particular marginal. On each $P_i(j)$, the aggregation has been computed on all attributes not present in the marginal j.

We denote as $Y_{A_1 A_2 \ldots A_n}$, the value of attribute Y for the specific combination of the A_i attributes (d of those are specified by the grid query and the remaining $n - d$ from the hierarchy). We denote as $\hat{Y}^{(t)}_{A_1 A_2 \ldots A_n}$ the estimate of the value of $Y_{A_1 A_2 \ldots A_n}$ during the t-th iteration of the algorithm. IPF starts the reconstruction by initializing a d-dimensional grid, G, of size $d_{i_e} - d_{i_s} + 1$ per dimension i, to one. We refer to each element of the d-dimensional grid as a *cell*. In addition, it computes the k marginals in S for the initialized grid G. Let $\hat{P}^{(t)}_i(j)$, denote the marginals computed from G in the t-th iteration of the algorithm. Denote $\hat{P}^{0}_i(j)$, the marginals after the initialization.

At each iteration, IPF loops over the k marginals $j \in S$, and all grid cells, $l_1 \in [d_{1_s}, d_{1_e}], \ldots, l_n \in [d_{d_s}, d_{d_e}]$, and adjusts the values of the grid cells according to the formula

$$\hat{Y}^{(t+1)}_{l_1 l_2 \ldots, l_d} = \hat{Y}^{(t)}_{l_1 l_2 \ldots l_d} \frac{P_i(j)}{\hat{P}^{(t)}_i(j)}, \tag{3.3}$$

where $1 \leq i \leq k$.

This procedure is guaranteed to converge to the maximum entropy estimation of P given the specified collection of marginals. The estimates converge in a monotonic fashion, which allows the application of a stopping rule. Commonly, we choose to terminate the iterations when the change in each individual estimate becomes smaller than some user specified δ

value. For all the experiments presented in this chapter (except when we evaluate the effect of parameter δ on the algorithm) we set δ to 10% of the median of the values designated by Q. We chose the median because it is not affected by any extreme values, and it can be efficiently computed [MRL99] and stored in the DBMS. In addition, as we show in the experimental section, the algorithm is not very sensitive to the value of δ, making the specific choice less important. A skeleton of the IPF algorithm is given in Figure 3.2.

Objective: Compute the maximum entropy estimates $\hat{Y}_{A_1 A_2 \ldots A_d}$ of a given d-dimensional dataset $Y_{A_1 A_2 \ldots A_d}$.
Input: A set of k marginals $P_i(j), i = 1, \ldots, k$ and a stopping threshold δ.
Output: The maximum entropy estimates $\hat{Y}_{A_1 A_2 \ldots A_d}$

```
1 let step t:=0;
2 initialize Ŷ_{A1 A2 ... Ad};
3 do {
4     for i:=1 to k {
5         ∀ l1 ∈ [d1s, d1e], ..., ld ∈ [dds, dde]
6         Ŷ^(t+1)_{l1 l2 ... ld} = Ŷ^(t)_{l1 l2 ... ld} P_i(j) / P̂_i^(t)(j)
7         for m:=1 to k {
8             update P̂_m^(t+1)(j) based on Ŷ^(t+1)_{l1 l2 ... ld};
9         }
10        t++;
11    }
12    let maxChange:=max{ |Ŷ^(t)_{A1 A2 ... Ad} − Ŷ^(t−k)_{A1 A2 ... Ad}| },
                          over all l1 ∈ [d1s, d1e], ..., ld ∈ [dds, dde];
13 } while maxChange≥ δ;
```

Figure 3.2: The IPF algorithm.

An important issue is which marginals to choose in order to use them for the estimation process. Clearly, different sets of marginals will yield different estimates. This will, in turn, result in different estimation errors. Another, equally important, consideration factor, is the space required to store the marginals. If the gain in the approximation accuracy is negligible when using marginals of a high order, then we may as well restrict ourselves to a cheaper alternative that has almost the same benefit.

At this point we know, based on Theorem 3.2, that when we use marginals of the same

order for the estimation procedure, then the higher the order of the marginals is, the more accurate the estimation becomes. Nevertheless, what we would really need is a measure of the relative benefit of using different sets of marginals. Note, that we should be allowed to choose marginals from different orders as well. This measure would serve as a means of comparison among the available choices of marginals, by taking into account both the space required to store each alternative, and the estimation accuracy that it achieves. Ideally, we would like to be able to compute this measure based only on the marginals.

Unfortunately, no measure with the desired properties is known. It turns out that in order to evaluate the available choices we have to turn our attention to the detailed data that produced the marginals, and try out the different alternatives [BW00]. In the absence of such a measure that can effectively quantify the relative merit of using alternative sets of marginals, for the rest of this thesis we restrict our attention to marginals of the same order. In Chapter 6, we discuss further some of the issues that arise when we have to make a choice among different sets of marginals.

3.4.2 An Example

We explain the operation of the algorithm with an example. For illustration purposes, we discuss an example in two dimensions, involving attributes A_1 and A_2. The generalization to higher dimensions is straightforward.

Assume that we have the dataset depicted in Figure 3.3(a), and we want to focus on the highlighted subset. Figure 3.3(b) shows the particular subset along with its marginals. The IPF algorithm starts by initializing the estimates $\hat{Y}^{(0)}_{l_1 l_2} = 1, l_1 = 2, \ldots, 4; l_2 = 1, \ldots, 4$ (Figure 3.3(c)). Then, in the subsequent steps, it fits the marginals one by one. First, it fits $P_1(A_1)$ using the formula $\hat{Y}^{(1)}_{l_1 l_2} = \hat{Y}^{(0)}_{l_1 l_2} \frac{P_1(A_1)}{\hat{P}^{(0)}_1(A_1)}, \forall l_1 \in [2, \ldots, 4], l_2 \in [1, \ldots, 4]$ and we obtain the table of Figure 3.3(d). Then, it fits marginal $P_2(A_2)$ using the formula $\hat{Y}^{(2)}_{l_1 l_2} = \hat{Y}^{(1)}_{l_1 l_2} \frac{P_2(A_2)}{\hat{P}^{(1)}_2(A_2)}, \forall l_1 \in [2, \ldots, 4], l_2 \in [1, \ldots, 4]$. The result is depicted in Figure 3.3(e), and this is the final set of estimates. Indeed, if we run one more iteration of the algorithm the elementary cell estimates will not change. Therefore, the condition at the end of the main loop of the algorithm will be satisfied, and the procedure will terminate.

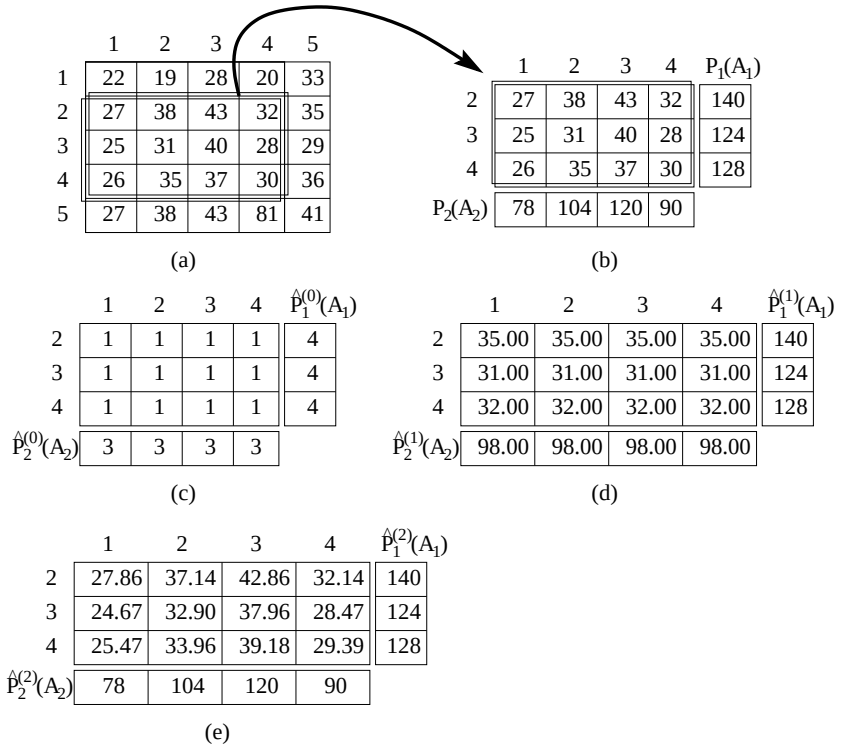


Figure 3.3: An example of applying the IPF algorithm. Figure (a) shows the whole dataset, and the portion of it that we wish to estimate. In (b) we have the subset we focus on, and its two corresponding marginals. The initialization step is depicted in (c). Figures (d) and (e) show how the algorithm fits the two marginals one at a time.

3.4.3 Algorithmic Complexity and Implementation Details

The IPF algorithm requires as input the marginals corresponding to a specific query Q, and then iterates over a grid G, which is the same size as the result set of Q. We can safely assume that the marginals fit in main memory, but this may not be true for G.

If the available memory is large enough to hold the grid G then the algorithm only needs to read from disk the marginals[2]. All the subsequent operations take place in main memory. Taking into consideration the large sizes of memory that are commonplace nowadays, we expect that the algorithm will be able to provide fast answers to a significant number of queries by operating on memory-resident data.

In the case where G does not fit in memory, the algorithm has an increased I/O cost. During each iteration it makes k passes over G, where k the number of marginals that we

[2]As we will show in Section 3.5, this involves just a selection query to the DBMS storing the underlying data.

use for the estimation. After having computed the estimates based on a single marginal for all the values in G, the algorithm has to make one more disk pass over G in order to calculate the new estimated marginals. In our implementation, we managed to cut the number of passes in *half* by incorporating the update of the estimated marginals with the computation of the base values, reducing the cost of the algorithm to $[kt]$ passes over G, where t is the number of iterations. This called for efficient indexes on the marginals, which were implemented using hashing techniques. The results that we report in the experimental section illustrate the worst case scenario, where the dataset does not fit in main memory.

The number of iterations the algorithm performs until convergence is achieved is usually small. Typically, six or fewer iterations are enough to get an accurate estimate, even for data sets of large dimensionality. In addition, since the algorithm is guaranteed to converge to the final estimates in a monotonic fashion, the user can stop the procedure when a sufficient number of significant digits of the estimates appear to have converged. Even though the parameter δ (which controls the accuracy of the estimation) is user defined, the above approach can still result in substantial savings in the computation time, compared to the case when the stopping criterion is only based on δ. An example of such a case is when the user is interested in a subset of the values that are being estimated, and for which the desired accuracy has already been achieved. Since this is a user interface issue, we do not pursue it further in the experimental section.

3.4.4 Quality Guarantees for the Approximation

The reconstruction process is only dependent on the *marginals* of the base data. This implies a significant reduction in the available information from which the base data will be estimated. For various applications, being able to provide error bounds for the reconstruction of individual values is imperative. For this procedure we can safely assume that, at the time of the computation of the aggregates, the original data are still available.

Let W be the set of grid queries of interest. One approach to provide error bounds for each query in W would be the following: estimate the values of each query in W, while the original data are still available, compute the largest difference (between the actual and the estimated value) for each query in W, and store them separately. This would incur a storage overhead of $O(|W|)$. More formally, let us denote by Y_i the original value of some cell i, and with $\hat{Y}_i$ its estimated value. We can use the absolute difference $d_i = |\hat{Y}_i - Y_i|$

in order to provide an upper bound for the error. Assuming that a specific grid query encompasses N cells from the base data, the upper bound can be calculated using the formula[3] $N \cdot \max_{1 \leq i \leq N}\{d_i\}$. Thus, the total error for the query is not going to be greater than N times the largest individual cell error.

The above error bound provides an indication of the accuracy of the reconstruction. Yet, some applications may require tighter quality guarantees. In order to provide such guarantees, we introduce the following approach: store a number k (user defined) of the largest estimation errors for each query in W. Given a query in W that involves a number of the cells whose correct values have been explicitly stored, the reconstruction algorithm uses these correct values, and thus induces no error for the specific cells. Since we have chosen to store the cells with the largest errors, the overall error for the query will be dramatically reduced. If the error bound per query should be specified by the user, we can choose k (the number of values to store) such that the overall error of reconstruction satisfies the error bound.

In many cases the number of cells for which the approximation is really poor will be relatively small. Therefore, the number of values that need to be materialized separately will be small as well. In the experimental section, we evaluate this argument, and we present graphs depicting the distribution of errors for the real datasets we used. As will become evident from the graphs, only a minor percentage of cells exhibit high errors, and thus, can be efficiently stored, with only a small storage overhead.

3.5 Using IPF

In the following sections, we discuss issues related to the applications of IPF, both for query answering and knowledge discovery.

3.5.1 Query Answering

Given a d-dimensional grid query Q the algorithm has to determine the order of the marginals to use for reconstruction as well as the ranges of these marginals pertinent to Q. In light of Theorem 3.2, marginals of order $d-1$ will produce the most accurate esti-

[3] We assume that the error metric is the Root Mean Square Error. Similar arguments hold if we choose other error metrics as well.

mate. In general, using marginals of order k with IPF will yield a more accurate estimate than using marginals of order $k-1$. In the experimental section, we present graphs exploring the time and accuracy tradeoffs related to this choice.

Assuming one decides to use the marginals of order k, the exact marginals relevant to Q have to be determined. A simple rewriting of Q can produce the $\binom{d}{k}$ marginals of order k that should be queried in order to retrieve the values for IPF to operate on. Any grid query on $r(A_1, \ldots, A_n)$ where $A_1, \ldots A_d$ are dimension attributes and $A_{d+1} \ldots A_n$ define hierarchies on them, can be expressed in SQL as:

SELECT $A_1, A_2, \ldots A_d$

FROM r

WHERE $A_{d+1} = a_1$ and $\ldots A_n = a_{n-d}$,

where the values $a_1, \ldots, a_{n-d}$ designate the multidimensional range of Q. Let $F = \{A_1, \ldots, A_d\}$. Then, the marginals of order $d-1$ relevant to the reconstruction of the grid query can be retrieved by issuing the following SQL query:

SELECT $F - A_i$

FROM r

WHERE $A_{d+1} = a_1$ and $\ldots A_n = a_{n-d}$,

for $1 \leq i \leq d$. This will give us all the $\binom{d}{d-1} = d$ marginals of order $d-1$. Similarly, the marginals of order $d-2$ relevant to the reconstruction of the grid query can be retrieved by:

SELECT $F - \{A_i, A_j\}$

FROM r

WHERE $A_{d+1} = a_1 \ldots A_n = a_{n-d}$,

for $1 \leq i < d$, $i < j \leq d$. Reasoning similarly, we can construct the expressions for the choice of marginals of any order.

3.5.2 Mining Interesting Patterns

In the case that the base data are available, the proposed reconstruction technique has a different utility. Maximum entropy reconstruction from a number of marginals is performed

based on the assumption that the marginals of interest are pairwise independent. By reconstructing the data and comparing them with the base data, the validity of the pairwise independence assumption can be tested. Any data value that violates the pairwise independence assumption, will induce a larger reconstruction error than one that does not. This provides an automatic way to reason about the underlying correlations among attributes.

For example, if the volume of sales of redtab Levi's jeans in Queens incurs the largest estimation error among all sales of Levi's in the cities of NY state, we know that the volume of sales in Queens represents the strongest violation of the assumption that sales in NY state are independent of the city. We term such values *deviations*, because they deviate from the estimation model. Such values can potentially be of great interest to the analyst since they record values out of the norm. This property can be utilized during grid query execution, since we can automatically identify and report values of potential interest to the analyst.

The basis for identifying a specific value as a deviant can be a measure of the distance between the actual and the estimated value, i.e., the estimation error, such as absolute difference. However, this metric may not always produce high quality results. An example of this case would be a dataset with values drawn from a uniform distribution. Then, most of the values in this dataset would qualify as deviants, which is not correct. In order to remedy this situation, we can use a formula which normalizes the estimation error of a value with respect to the standard deviation σ of all the estimation errors returned by the algorithm for the underlying dataset

$$s = \frac{|\hat{Y}_i - Y_i|}{\sigma},$$

where with Y_i we denote the original value of cell i, and with $\hat{Y}_i$ its estimation. Then, we choose a cutting threshold for s, that can effectively differentiate between the normal perturbations in the dataset and the large deviations. The above technique splits the sorted set of deviations into two regions: it assigns the statistically large deviations to the first region, and the rest to the second one. We refer to the boundary point between those two regions as the *cutoff point*. A commonly used threshold is $s = 2$, which will prune 95% of the approximation errors as trivial, leaving only the largest 5% for consideration (the values follow from the properties of Normal distributions). The system can subsequently sort those deviating values, and pick the top-k among them. In the experimental section, we present

graphs that visualize the deviations determined by the algorithm, for both synthetic and real datasets.

3.6 Experimental Evaluation

In the following subsections, we present the experiments we used to evaluate the performance of the IPF algorithm.

First, we describe the real and synthetic datasets we employed in our evaluation. Then, we assess the ability of the algorithm to make accurate estimates of some unknown multidimensional distributions. Finally, we test another application of the IPF algorithm, namely mining interesting patterns and identifying deviations.

3.6.1 Description of Experiments

In order to test the IPF algorithm, we used both synthetic and real datasets. The synthetic datasets are produced by sampling uniform and Gaussian data distributions.

Uniform: We produced datasets of dimensionalities 2, 3, and 4. For each of the above datasets of different dimensionality, the values were drawn uniformly from the range $[0, 10]$ (*uniform10*), $[0, 100]$ (*uniform100*), and $[0, 1000]$ (*uniform1000*). The size of the datasets varied from 1,000 to 20,000 tuples. The statistical properties of those datasets are reported in Table 3.1.

Gaussian: We produced datasets of dimensionalities 2, 3, and 4. The values were sampled from independent Gaussian distributions. For σ we chose three different values that altered the distribution of the values in the data space. In the experiments we refer to these datasets as *gauss_small*, *gauss_medium*, and *gauss_large*. Once more, the size of the datasets varied from 1,000 to 20,000 tuples. The statistical properties of those datasets are reported in Table 3.1.

We also experimented with three Gaussian datasets which had additional random noise coming from a uniform distribution. The first dataset has small variance, while the second one has large variance. The third dataset was derived from a mixture of two multidimensional Gaussian distributions. We refer to these datasets as *gauss_flat*, *gauss_bell*, and *gauss_combined* respectively. A high level description of how the mix-

ture of Gaussian datasets were generated is shown in Figure 3.4. The statistical

Objective: Construct a $d-$dimensional dataset that follows a mixture of Gaussians distribution.
Input: A pair of values (μ_j^k, σ_j^k) for each dimension k, and for each of the Gaussian distributions j that we want to be present in the dataset.
Output: The $d-$dimensional data values.

```
1 let {x_1, ..., x_d} be the d dimensional attributes;
2 let N be the number of points dropped in the multidimensional grid;
3 for i:=1 to N {
4     for j:=1 to number of Gaussians in the dataset {
5         for k:=1 to d
6             let x_k:=gauss(μ_j^k, σ_j^k);
7         increment the value of position (x_1, ..., x_d) in the dataset by one;
8         }
9     }
10 /* we then add some uniform noise on top */
11 for all values in the dataset {
12     withprobability 0.01 add to the value of position (x_1, ..., x_d) a value drawn
           uniformly at random from the range [0,mean of dataset];
13     }

14 procedure gauss(μ, σ)
15     let r be a value drawn from a Gaussian distribution with mean μ and standard
           deviation σ;
16     return (r);
```

Figure 3.4: The algorithm for generating the Gaussian synthetic datasets.

properties of the datasets are reported in Table 3.1.

Real: The first of the real datasets, *calls* and *calls3*, are derived from AT&T proprietary data. They represent aggregated telephone calls in certain regions of North America over time. They are 2- and 3-dimensional, and their size is 10,000 tuples. The dimension attributes are time, location of the call, and customer type.

Finally, we used *census*, a dataset from the U.S. Census Bureau, containing information about the age, education, command of English and number of children of individuals. It is a 4-dimensional dataset from which we extracted instances of 10,000-50,000

dataset	*min*	*max*	*mean*	*std.dev.*	*skew*
uniform10	0.0	10.0	5.01	2.88	0.01
uniform100	0.0	100.0	50.11	28.78	0.01
uniform1000	0.0	1000.0	501.14	287.84	0.01
gauss_small	38.0	115.0	73.46	10.26	0.11
gauss_medium	4.0	100.0	42.90	16.24	0.40
gauss_large	0.0	104.0	15.60	18.34	1.65
gauss_flat	9.0	76.72	28.15	5.80	0.40
gauss_bell	0.0	72.14	20.72	12.45	0.36
gauss_combined	0.0	106.68	24.66	13.57	0.53

Table 3.1: The statistical properties (min, max, mean, standard deviation, and skew) for the synthetic datasets used in the experiments.

tuples by uniform random sampling.

Table 3.2 summarizes the statistical properties of the real datasets we used in our experiments.

dataset	*min*	*max*	*mean*	*std.dev.*	*skew*
calls	1.0	729.00	18.01	37.79	5.37
calls3	0.0	729.00	9.01	9.32	22.77
census_10K	1000.00	196623.00	24735.73	23449.87	3.67
census_20K	1000.00	196623.00	24894.82	23412.84	3.63
census_30K	1000.00	196623.00	24751.72	23278.56	3.66
census_40K	1000.00	196623.00	24718.29	23211.15	3.62
census_50K	1000.00	196623.00	24817.64	23331.80	3.61

Table 3.2: The statistical properties (min, max, mean, standard deviation, and skew) for all the real datasets.

The error metric that we report in the experiments is the *Root Mean Square Error (RMSE)*, defined as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Y_i - \hat{Y}_i)^2}{N}},$$

where Y_i represents the original values in the dataset, $\hat{Y}_i$ the corresponding estimated values, and N is the total number of values in our dataset. In the following experiments, we estimate the values of each dataset and measure the error of the estimation for all N values of the dataset.

Note that all the above datasets also include a measure attribute. Therefore, the total number of attributes for the datasets we used ranges from 3 to 5.

3.6.2 Exploring the Properties of the Algorithm

In this section, we present experiments concerning the choice of the parameter δ, the run-time of the algorithm, and the property stated in Theorem 3.2, namely the fact that higher order marginals produce more accurate estimates.

The following experiments examine the effect of the parameter δ on the performance of the algorithm. We ran the algorithm with δ varying from 5% to 15% of the median (see Section 3.4.1) for several synthetic and real datasets. Table 3.3 shows that the error does

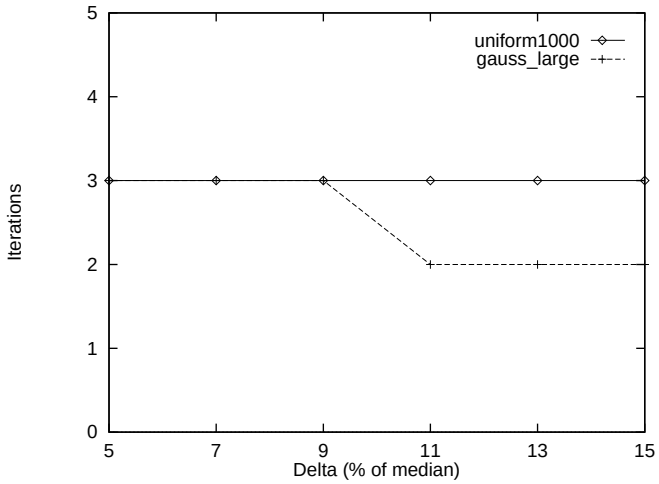
δ (% of median)	*root mean square error*				
	gaussian	**uniform**	**census_10K**	**census_30K**	**census_50K**
0.001	2.072	227.02	16889.76	18184.32	18564.82
5	2.072	227.09	16892.78	18185.64	18565.95
15	2.072	227.09	16895.72	18187.71	18567.60

Table 3.3: The error for the various datasets as a function of the parameter δ.

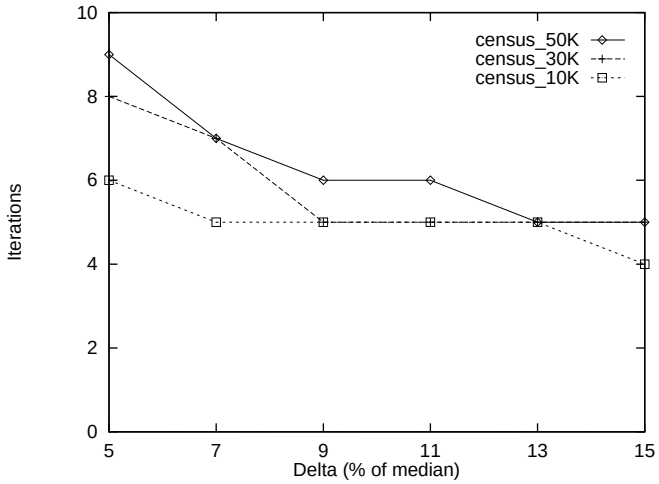
not vary much when the value of δ increases. Even if we set δ to a very small value (i.e., $\delta = 0.001$) the benefit we get in the reduction of the error is not significant (results in the first row of the table). Since the choice of δ is not crucial for the error, we would like to pick a value that results in a small number of iterations for the least possible error. The graphs in Figure 3.5 depict the number of iterations needed for each value of δ, when the IPF algorithm is applied to the entire dataset. The largest variations in the number of iterations are observed in the datasets with the highest skew (Figure 3.5(b)). Yet, even for those datasets, setting δ to 10%, which is the setting we used throughout our experiments, proves to be a good choice. We expect that for most of the datasets this choice will lead to good performance of the algorithm.

We also performed some experiments to test the effect of dataset size on the convergence of the algorithm. These experiments indicate that there is no correlation between the dataset size and the number of iterations that the algorithm needs to perform in order to converge.

Figure 3.6(a) shows how the run-time of the algorithm changes when the dataset size increases. The graph demonstrates the outcome of experiments for datasets of different

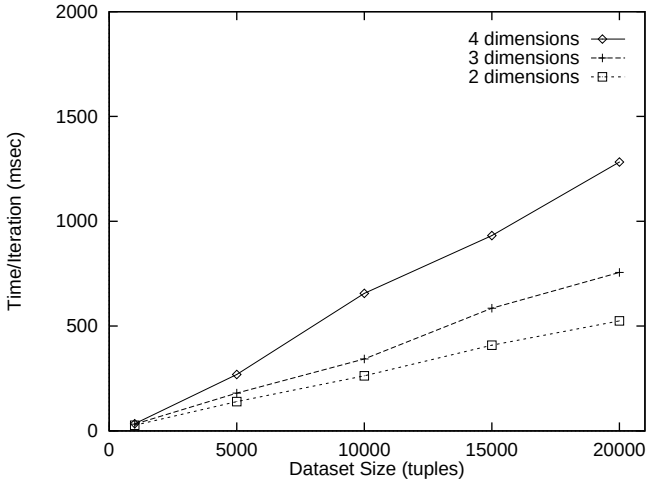


(a) Iterations vs Delta, synthetic datasets

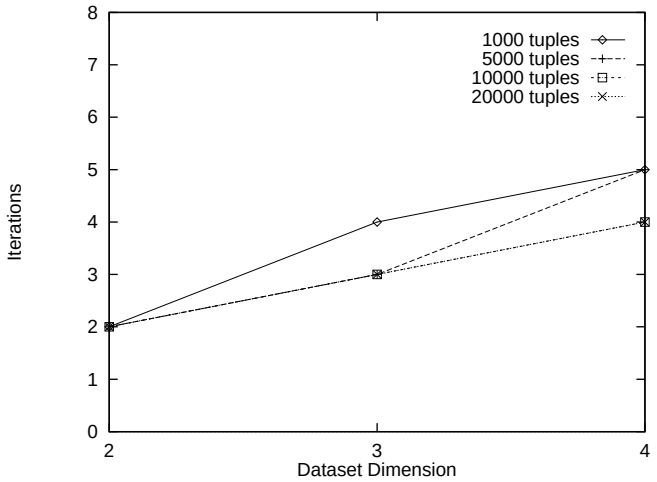


(b) Iterations vs Delta, *census* datasets

Figure 3.5: The effect of varying the parameter δ on the number of iterations. All datasets are 4-dimensional. In every case the marginals of the highest order were used.



(a) Time vs Dataset Size



(b) Iterations vs Dataset Dimensionality

Figure 3.6: The effect of dataset size on the run-time of the algorithm (time per iteration), and of dataset dimensionality on the number of iterations. The parameter δ is set to 10% of the median, and we use the marginals of the highest order. For these experiments we used the uniform datasets.

dimensionalities. As expected, there exists a linear relationship between the size of the dataset and the run-time of the algorithm. Moreover, when dimensionality increases, the number of marginals that the algorithm uses increases as well. This explains the steeper slopes for the curves in Figure 3.6(a) for the higher dimensions. As Figure 3.6(b) depicts, the number of iterations increases with the dimensionality of the dataset. For these experiments, we used the marginals of the highest possible order, and set δ to 1% of the median in order to exaggerate the differences.

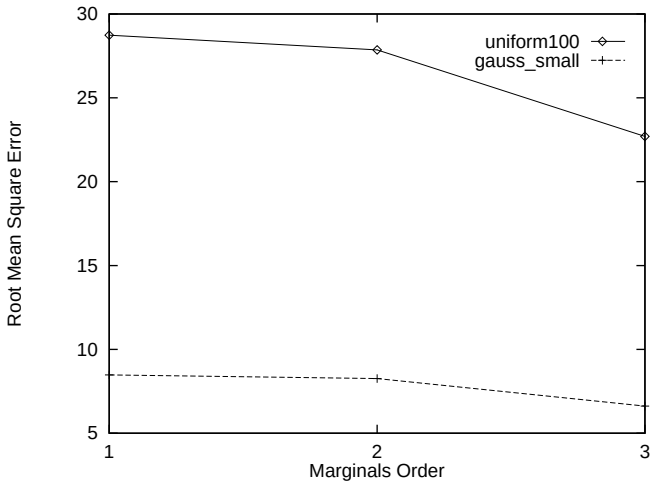
These results on the scalability of the algorithm demonstrate that this approach can be effectively used by the analyst in real time, and in an interactive fashion, even for queries that do not fit in main memory.

Figure 3.7 depicts the experimental verification of Theorem 3.2. We used 4-dimensional datasets, and measured the error of the estimation when the algorithm operates with marginals of orders 1 to 3. Figure 3.7(a) is an illustration of the fact that when we use higher order marginals for the reconstruction of same dataset, the error is diminishing. The large difference in the errors shown in the graph of Figure 3.7(a) is explained by the fact that the uniform dataset has much larger variance than the Gaussian one (see Table 3.1). It is interesting to note here that, for the datasets we used, the relative benefit of employing marginals of higher order is increasing. The reduction in error is larger in moving from order 2 to order 3 marginals, than it is for moving from order 1 to order 2 marginals. The greater accuracy that we are gaining by using marginals of high order comes at the expense of time (and, of course, space). The run-time of the algorithm increases with the order of the marginals (Figure 3.7(b)).

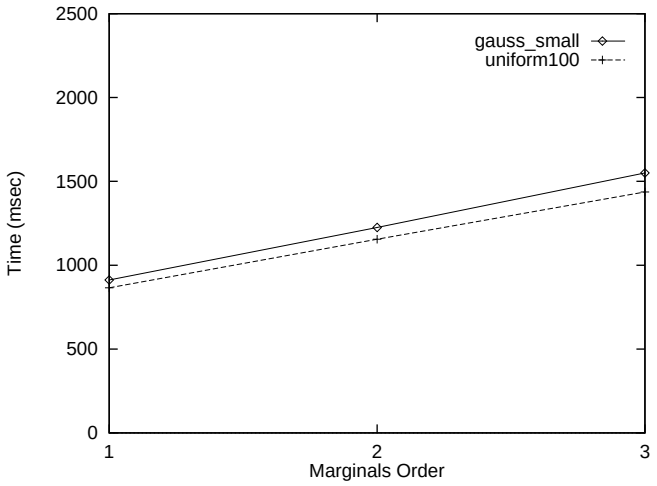
3.6.3 Evaluating the Accuracy of Reconstruction

The experiments described in this section investigate the behavior of the algorithm when reconstructing the detailed values of a dataset.

The graph in Figure 3.8(a) shows how the error changes when the dataset size increases, for the three uniform distributions. All the datasets have three dimensions. As is evident from Figure 3.8(a), the error of reconstruction is related to the variance of the underlying dataset and it increases as the variance increases. By increasing the size of the dataset, the error increases (but not dramatically) as more values are included in the computation of the error. The next graph, Figure 3.8(b), depicts how the dataset dimensionality affects

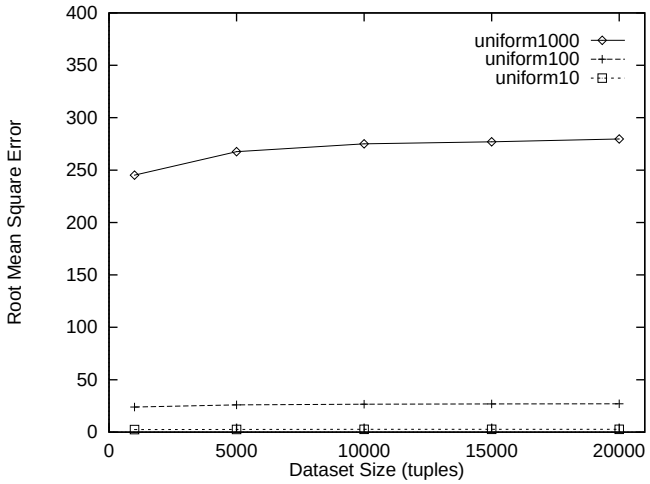


(a) Error vs Marginal Order

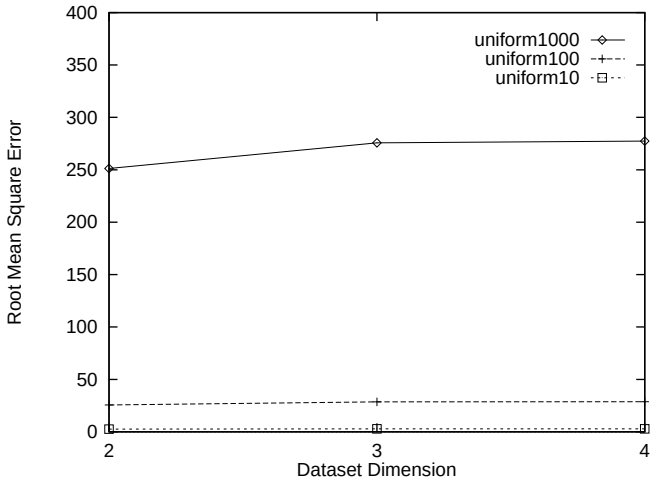


(b) Time vs Marginal Order

Figure 3.7: The effect of the order of the marginals used on the error of reconstruction and the run-time of the algorithm. The datasets used in these experiments are *uniform100* and *gauss_small*.



(a) Error vs Dataset Size, used marginals of order 2

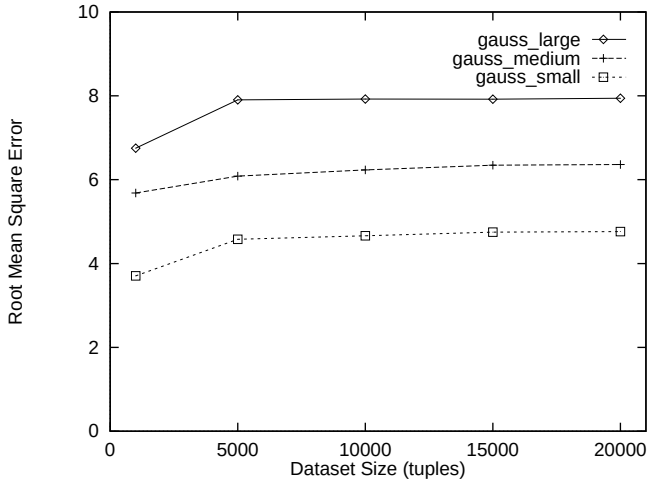


(b) Error vs Dataset Dimensionality, used marginals of order 1

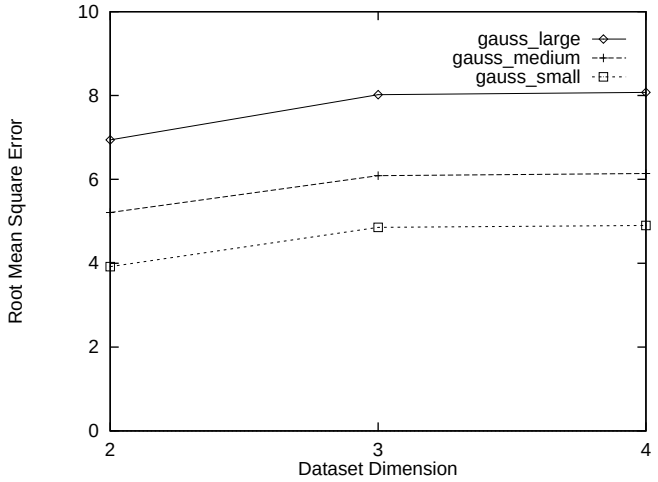
Figure 3.8: The effect of dataset size and dimensionality on error, for uniform datasets. Three uniform distributions with different mean values are represented in the graphs.

the accuracy of the estimations. During this experiment, the algorithm was applied to the marginals of order 1, which is the highest order common to all datasets. It is evident that the reconstruction error increases with dimensionality; however, the increase seems correlated to the variance of the underlying dataset, since the increase of the error as the dimensionality increases is small.

Figure 3.9 depicts the way error changes with respect to dataset size and dimensionality, in the case of Gaussian datasets. The results we get are similar to those in the experiment with uniform datasets.



(a) Error vs Dataset Size, 3-dimensional datasets



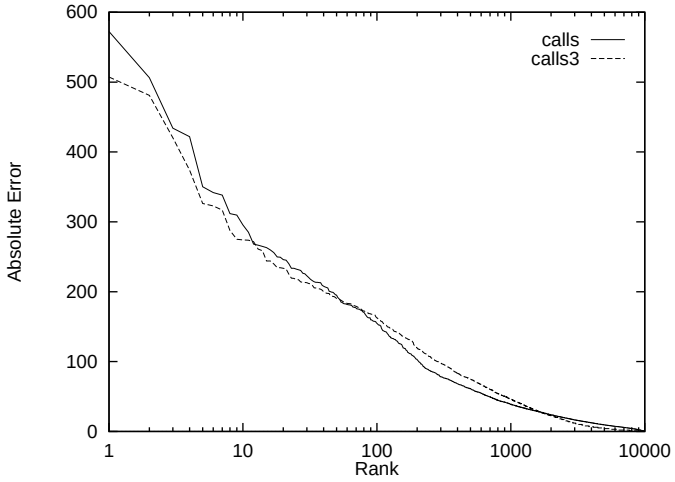
(b) Error vs Dataset Dimensionality, 10000 tuples

Figure 3.9: The effect of dataset size and dimensionality on error, for Gaussian datasets. Three Gaussian distributions with different sigma value are represented in the graphs.

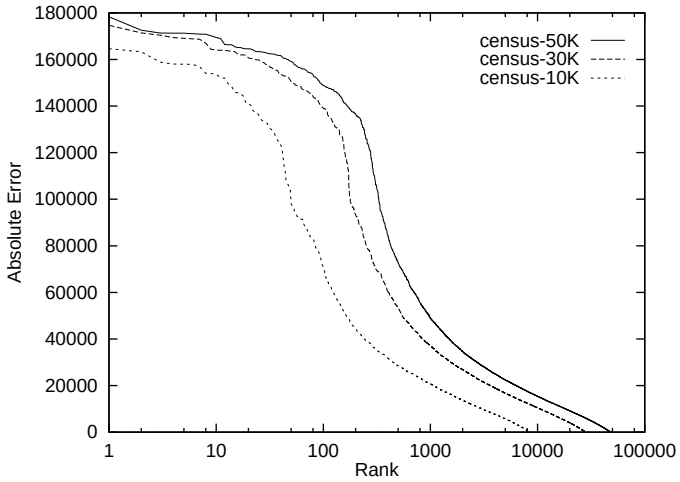
3.6.4 Reconstruction with Quality Guarantees

In the following experiments, we assess the benefit of providing error bounds. The method we propose (as discussed in Section 3.4.4) explicitly stores a small number of deviating values, which are subsequently used during the reconstruction phase to diminish the error.

In the first set of experiments, we explore the distribution of the size of the estimation errors (i.e., the absolute error between the real and the estimated value for an individual cell). The graph in Figure 3.10(a) depicts the distributions for the datasets *calls* and *calls3*



(a) *calls* Datasets.



(b) *census* Datasets.

Figure 3.10: The distribution of the absolute error for the real datasets.

after sorting into decreasing size. Both curves indicate that the error sizes follow a skewed

distribution, with a fraction of the errors having large values. This fact indicates that the choice to store the largest estimation errors as extra information is likely to pay off during reconstruction. Figure 3.10(b) shows the same graph for different sizes of the *census* dataset. The results show that, for all the sizes, the distribution of the errors is skewed.

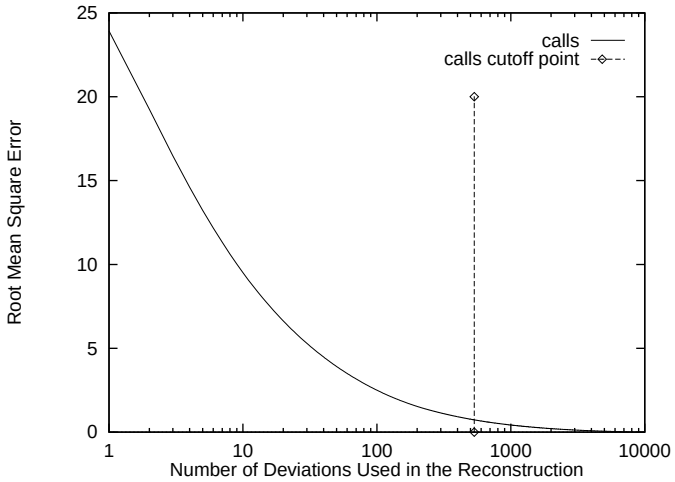
The next experiments evaluate the relative benefit of storing a number of deviating values per query in order to guarantee a specified reconstruction error bound. Figure 3.11 shows the error of the reconstruction as the number of deviations that are stored increases from 0 to the size of the dataset. The user may choose between those two ends according to the application requirements for quality guarantees, and the space restrictions on the number of extra values that the algorithm will use.

In every case, storing only a very small number of deviations is enough to dramatically decrease the error. For both real datasets we used, storing only the few largest deviations decreased the error by two orders of magnitude. As expected, at the other end of the spectrum, when the deviations for all the values are stored, we can use this information to achieve a perfect reconstruction.

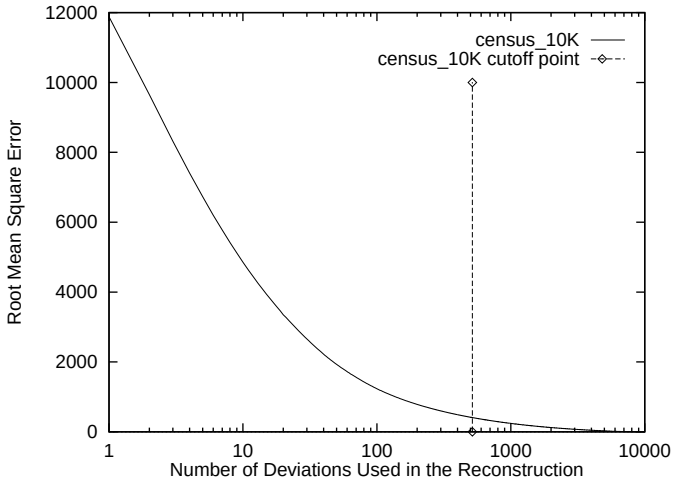
Note the role that the *cutoff point* can play in this situation (see Section 3.5.2). In the graphs, the cutoff point, is marked with a vertical line, and it can be used to determine the point (and subsequently the number of values to materialize) at which the relative benefit of storing additional deviating values becomes negligible.

In Figure 3.12, we use a logarithmic scale for the y-axis and a linear scale for the x-axis, to present the same graphs as before. In addition to the actual reconstruction error, we plot a theoretical upper-bound for the error, following the discussion in Section 3.4.4. Even though this bound is not tight, it may still be useful for certain kinds of applications. These graphs also enforce our argument about the *cutoff point*. It is clear that the *cutoff point* separates the initial region of dramatic decrease of the error from the plateau that follows. The added benefit of storing extra values from the latter region is quite small.

In the following experiments, we evaluate the tradeoff between the accuracy of the estimation and the space needed to achieve this accuracy. Figure 3.13 shows the reduction in the estimation error when we use marginals of successively higher order, and, in Figure 3.14, we show the increase in the space needed by these marginals. When we use marginals of higher order, the accuracy of the estimation increases. However, this increase in the accuracy comes at the expense of space, since the space needed by the marginals of higher

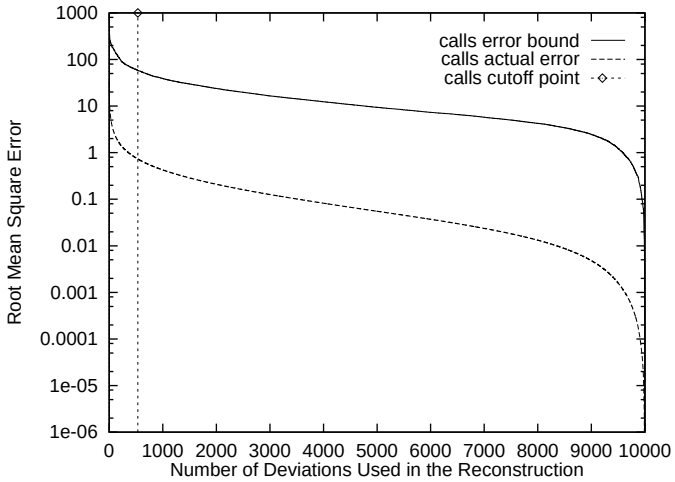


(a) *calls* Dataset.

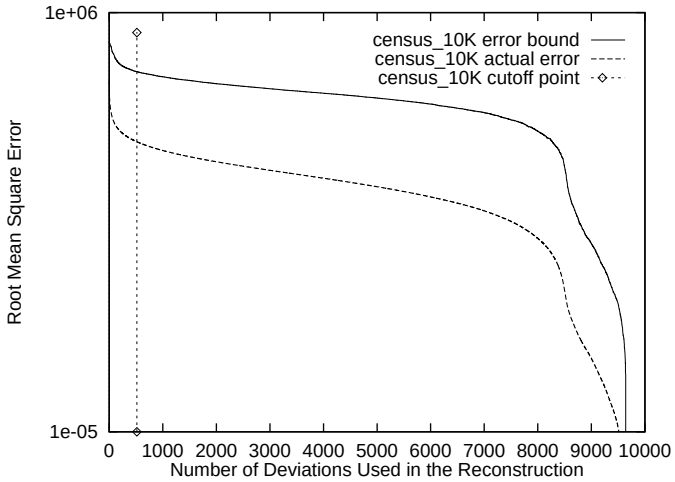


(b) *census_10K* Dataset.

Figure 3.11: The reconstruction error when a varying number of deviations is used by the algorithm.



(a) *calls* Dataset.



(b) *census_10K* Dataset.

Figure 3.12: The actual reconstruction error and an upper bound when a varying number of deviations is used by the algorithm.

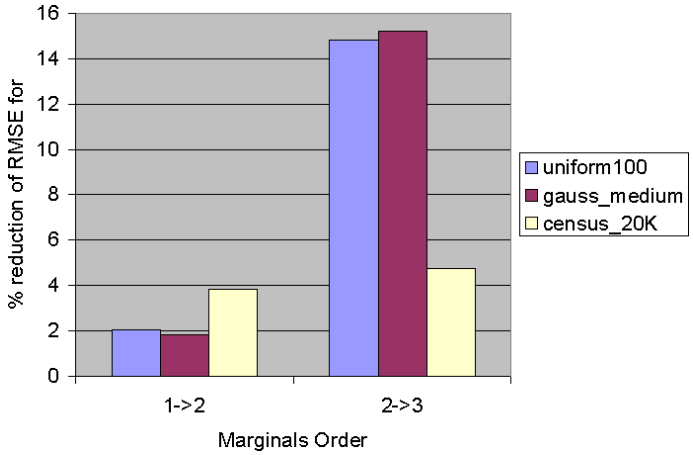


Figure 3.13: The percentage of the error reduction when we use the marginals of order $i+1$ instead of the marginals of order i, in the estimation process.

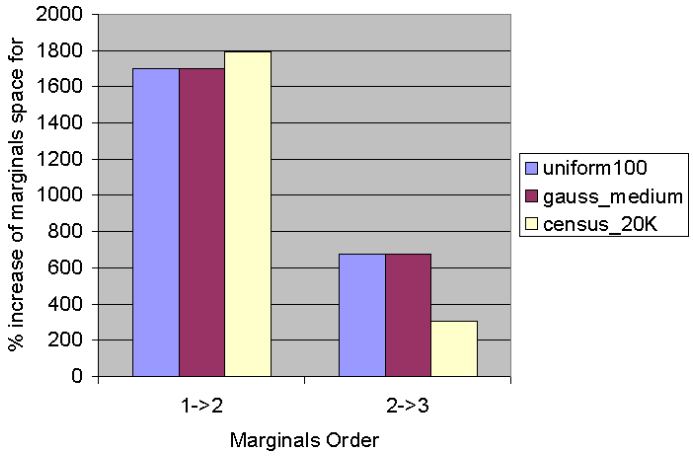


Figure 3.14: The percentage of the increase in the space occupied by the marginals, when we use the marginals of order $i+1$ instead of the marginals of order i, in the estimation process.

order increases as well. If we compare the graphs illustrated in Figures 3.13 and 3.14, we observe that in order to improve the estimation accuracy by a small amount, we have to use disproportionally more space. That is, there is small added benefit for the extra space needed by the marginals of higher order.

In the above experiments we reduced the estimation error by employing marginals of higher order. Another way of reducing the estimation error is to explicitly store a number of the largest deviating values. For the following experiments, we measure the reduction in the estimation error when we use the marginals of some order i, and a number of deviations. We ensure that the total space required by both the marginals of order i and the stored deviations, equals the space required by the marginals of order $i+1$. In this way, we can compare the benefit of explicitly storing some deviating values, against using marginals of higher order, for some fixed space. Figure 3.15 depicts the outcome of these experiments. We observe that in this case the reduction of the estimation error is considerable, and the reconstruction of the real values of the datasets is almost perfect.

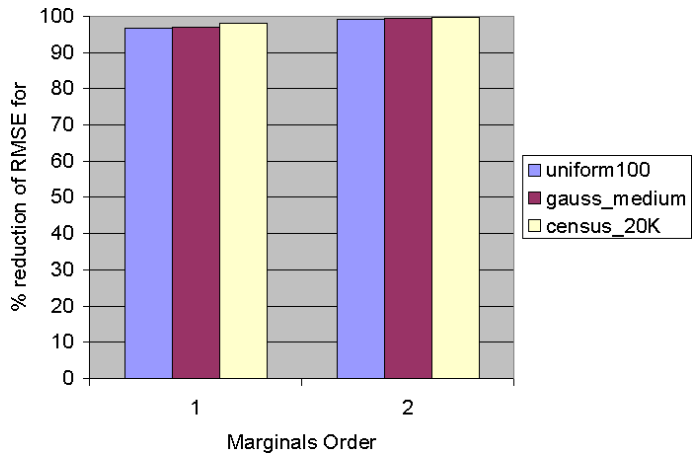


Figure 3.15: The percentage of the error reduction when, in the estimation process, we use the marginals of order i and a number of deviations, such that the total space equals the space needed by the marginals of order $i+1$.

Evidently, for the same amount of space, it is much more beneficial to store the marginals of some order i along with a number of the largest deviations, than to store the marginals of order $i+1$. Nevertheless, it is not always the case that we can explicitly store the deviating values. For example, consider the scenario of an online system, where only the marginals of some measures of interest are materialized, and the detailed values are not stored. Then,

we will have to use just the marginals for the estimation process.

3.6.5 Mining Interesting Patterns

We evaluate the ability of the IPF algorithm to mine the underlying general structure of the data and report any deviations with the following experiments with synthetic and real datasets. Note that the graphs we present involve datasets in two dimensions only, for illustration purposes.

Synthetic Datasets

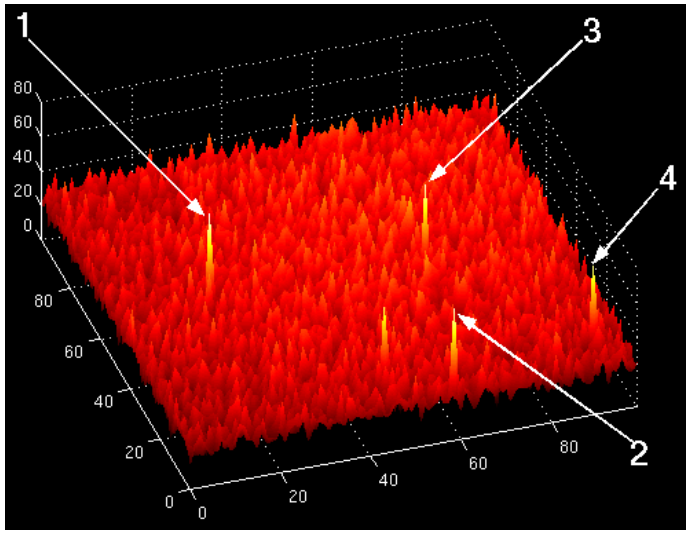
We produced two datasets (namely *gauss_flat* and *gauss_bell*) drawn from Gaussian distributions and then added some uniform noise on top, as described in Figure 3.4. These datasets are shown in Figure 3.16. The marginals created by the algorithm captured the general trends of the data, and the algorithm was able to report the values that deviate the most from the real distribution. The top-4 of these values for each dataset are presented in Table 3.4. Manual inspection of the results reveals that these are indeed the predominant

gauss_flat		*gauss_bell*		*gauss_combined*	
cell	diff.	cell	diff.	cell	diff.
(50,21)	47.34	(74,14)	56.80	(60,67)	62.78
(2,59)	47.27	(89,25)	44.01	(48,6)	58.33
(48,68)	42.15	(19,54)	42.74	(28,2)	52.26
(11,93)	41.24	(95,95)	36.98	(23,60)	36.81

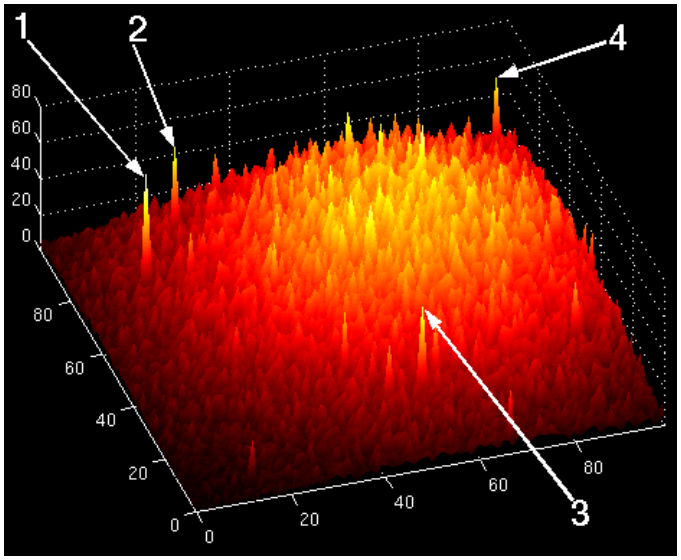
Table 3.4: The top-4 deviations reported for each of the three synthetic datasets. The metric used is the *difference* of the real value from the estimated.

deviations in the datasets.

The third synthetic dataset (*gauss_combined*) we tested is a combination of two multi-dimensional Gaussian distributions with different mean and sigma values, and some noise on top (generated as described in Figure 3.4). It is depicted in Figure 3.17(a). Once more, the algorithm correctly singled out the most significant deviating values (reported in Table 3.4). Note that the algorithm *does not merely identify global phenomena*, e.g., reporting the maximum value along a dimension. Instead, it takes into account the local neighborhood in which a particular value appears, and reports any incongruities therein. For example,

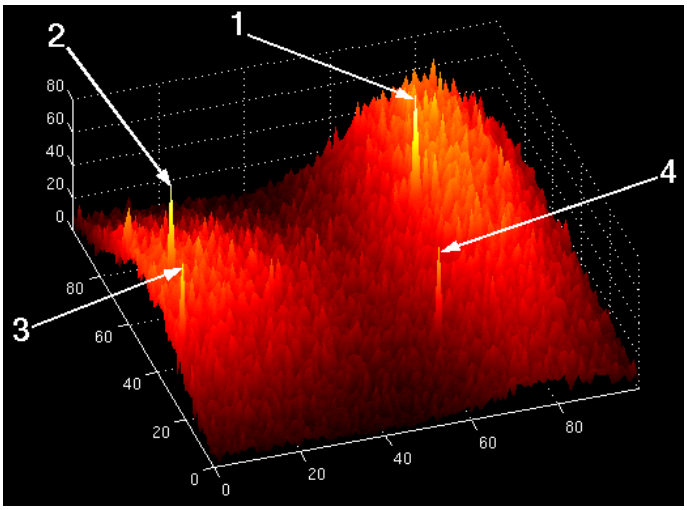


(a) *gauss_flat* Dataset

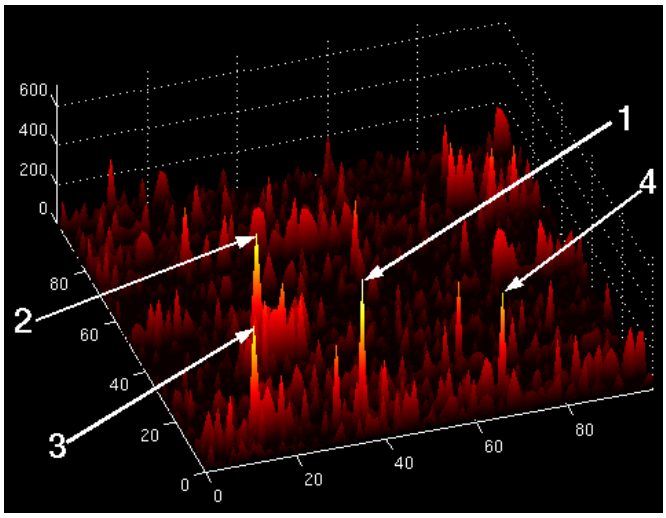


(b) *gauss_bell* Dataset

Figure 3.16: Illustration of the original datasets following Gaussian distributions with uniform noise added on top. The four largest deviating values are marked in the graphs.



(a) *gauss_combined* Dataset



(b) *calls* Dataset

Figure 3.17: Illustration of the two datasets where the four largest deviating values are marked. The dataset on the top is synthetic (combination of two different Gaussian distributions with uniform noise added on top), while the one at the bottom is the *calls* dataset from AT&T.

consider the deviation number 4 reported by the algorithm, in Figure 3.17(a). This value is not the maximum value of the dataset (its rank in the dataset is 116-th). Nevertheless, it is identified by the algorithm as a deviant, because it differs significantly from all the other values in its neighborhood.

Real Datasets

In the following experiments, we used the algorithm to find the most deviating values in two of the real datasets, the *calls*, and the *census_10K* dataset.

Figure 3.17(b) depicts the *calls* dataset along with the top-4 deviating values, which are also listed in Table 3.5. All the marked values are instances of unusually high volume of calls. This information is important to the analyst since it indicates exceptional behavior which can either be fraudulent, or signify special cases in the dataset.

calls	
cell	diff.
(7,37)	611.45
(38,24)	572.39
(7,13)	506.32
(7,68)	434.07

Table 3.5: The top-4 deviations reported for the *calls* datasets. The metric used is the *difference* of the real value from the estimated.

The outcome of the second experiment, with *census_10K*, cannot be graphically depicted, because the dataset is 5-dimensional. However, it is interesting to report some of the findings of the algorithm. The dimensional attributes of the dataset are age, command of English, number of children, and level of education, while income is the measure attribute. As expected, the above attributes are not independent. For example, the income tends to increase with age and with the level of education. Nevertheless, there exist values that do not follow these patterns. Among the top deviations are a middle-aged person with high level of education who earns less than 20K, a person with a PhD degree who earns merely 3K, and a 24-year old who earns 200K. These are certainly results that deviate from the norm, and therefore are interesting. (The reported deviations are among the top-30 returned by the algorithm. The rest of the deviations are not as interesting.)

Note that the algorithm is able to identify all the above results as interesting even though it has no domain knowledge, and it gets no user input.

3.7 Related Work

The principle of maximum entropy [CT91] has been successfully applied in different domains, including linguistics [CR99] [BPP96] and databases [FJS97a]. Faloutsos et al. apply maximum entropy in addition to other techniques, for one dimensional data reconstruction [FJS97a]. Our work generalizes the work of Faloutsos et al. to multiple dimensions.

There exists a sizeable bibliography in approximate query answering techniques [IP95] [PIHS96a] [JKM+98] [VWI98] [AGPR99] [SFB99] [BS97] [BW00]. Our approach is fundamentally different. Previous work focused on the problem of data reconstruction by constructing specialized summarized representations (typically histograms) of the data. We argue, that since there exist data that are already stored in an aggregated form in the warehouse, it is imperative to examine the quality of reconstruction one can attain from the aggregates.

The problem of identifying interesting values in a dataset is related to deviation detection. Arning et al. [AAR96] try to identify the subset of a database that is most dissimilar to the rest of the data. Other approaches discuss algorithms specialized to metric spaces that scale to large datasets [KN98] [KN99]. The drawback of these approaches is that the user is required to come up with the right selection of functions and parameters, which requires a great deal of effort. Our algorithm does not require such input, making the whole procedure less cumbersome and more robust. In that sense, our work is closer to the framework proposed by Sarawagi et al. [SAM98]. They describe an algorithm that mines the data in a data cube for exceptions. We should note that in certain cases the exceptions identified by this algorithm are the same as the deviations reported using our technique. This is because the log-linear models used in the above work produce the same model as the one derived using the maximum entropy principle, when the input is the same set of marginals [BFH75]. However, their method is computationally expensive, depends on the computation of the entire data cube, and cannot accommodate updates.

3.8 Conclusions

In this chapter, we considered the problems of using the aggregate information in order to provide approximate answers to queries and identify interesting values in multidimensional datasets. Each problem is of particular interest in the field of data analysis and approximate query answering respectively, especially since the volume of data stored in warehouses is huge. The techniques we discussed are based on the well recognized and widely applicable information theoretic principle of *maximum entropy*. We also proposed an extended framework that allows the user to choose quality guarantees for the reconstruction process. Finally, we presented an experimental study using both real and synthetic data, which explored the properties of the approach.

Chapter 4

Space Constrained Selection in Data Warehouses

So far we have assumed that there is enough space to store all the marginals needed to reconstruct each query in our workload. Now consider the case where we are allowed only a limited amount of space for storing the marginals. We can compute the space requirements of the marginals when they are materialized (either by computing them or by applying estimation techniques [SDNR96]). In this setting, we would like to materialize a subset of the marginals that satisfies the given space constraint, and at the same time is necessary for the reconstruction of the most important queries. (The set of aggregates needed to answer a specific query is determined by the reconstruction algorithm, as presented in Chapter 3.) The importance, or benefit, of each query can be manually set by the user, or it can be determined automatically by observing the occurrence frequency of the query in the system workload [ACN00] [LL02]. We discuss further this issue in Section 4.3.1.

In this chapter, we formally define the above optimization problem, and propose several algorithms for its solution. To the best of our knowledge, this is the first attempt to study solutions for this problem.

4.1 Contributions

The contributions we make in this chapter are as follows.

- We formulate optimization problems concerning the selection of *sets* of items that

under a space constraint yield the highest benefit, where benefits are associated with sets of items. This kind of optimization problem appears in various domains, and is very interesting in practice.

- We derive the complexity of these optimization problems, and propose several algorithms for their solution. Since there are no known polynomial-time approximation algorithms for these problems, we examine the use of known optimization principles in this context [GMW81]. Such principles include greedy, randomization, as well as optimization based on clustering.

- We explore the properties and special characteristics of the above techniques with an experimental evaluation. Our results illustrate the behavior of the algorithms under different settings, and highlight the benefits of each approach.

- Based on our analysis, we present lower bounds on the quality of the solutions produced by the algorithms. This offers insight into the operation of the algorithms, and provides a practical guide for selecting among the techniques proposed.

4.2 Outline

The outline of the rest of this chapter is as follows. Section 4.3 presents the formulation of the optimization problem, and Section 4.4 discusses algorithms for their solution. In Section 4.5, we present experimental results evaluating the performance and the utility of the proposed algorithms. Section 4.7 reviews the related work, and we conclude in Section 4.8.

4.3 Problem Formulation

In the following paragraphs, we establish the terminology used in the rest of the chapter, we present the formal statement of the problem, and we discuss its complexity.

Let *components* be the marginals that are available for use, and *objects* be the queries that can be estimated using the available marginals. When all the components required by an object are selected in the solution, we say that the object is *satisfied*. We assume that all benefit values are positive. Each object is associated with a benefit, which is claimed when

the object is satisfied. Each component is associated with a space requirement, which is the storage space it will occupy when selected. When a component is selected we say that it is *materialized.*

Example 4.1 *According to the terminology we introduced above, the problem of view selection for approximate querying in datacubes can be translated as follows. The queries that we want to answer are the objects, while the sets of marginals needed by the estimation algorithm are the components.*

We can construct a bipartite graph $G(U, V, E)$, where U is the set of objects, V is the set of components, and E the edge set of the graph. An edge exists between $u \in U$ and $v \in V$ if component v is required by object u. An example of the general form of the bipartite graph is shown in Figure 4.1.

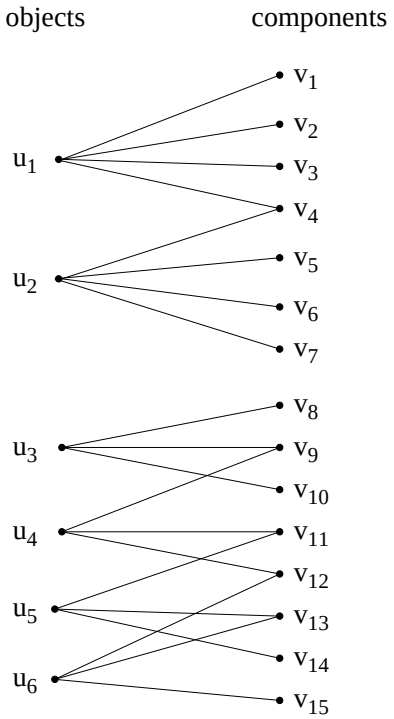


Figure 4.1: An example of the bipartite graph G.

Let $V[i], 1 \leq i \leq |V|$ be a binary vector having a 1 in position i if and only if we have selected component v_i for materialization, and $s(v_i), 1 \leq i \leq |V|$ be a function determining the space requirements of component v_i. Let $U[j], 1 \leq j \leq |U|$, be a binary vector having a 1

in position j if and only if all the components required by object u_j have been materialized. Each object u_j is also associated with a value $b(u_j)$, which specifies its benefit, and is a measure of its importance. Given a constraint W on the total space available for the components, we are interested in maximizing the total benefit of objects answered, while satisfying the space constraint. Then, the *Constrained Set Selection (COSS)* optimization problem can be stated as follows.

Problem 4.1 [The COSS Problem] *Maximize* $\sum_j U[j]b(u_j)$, *subject to the constraint that* $\sum_i V[i]s(v_i) \leq W$.

We show below that the integral *Knapsack* problem [GJ79] reduces to a special case of the *COSS* problem. Hence, according to the following lemma, the optimization problem is NP-Hard.

Theorem 4.1 *The* COSS *problem is NP-Hard.*

Proof: We will show that the integral *Knapsack* problem reduces to *COSS'*, a special case of *COSS*. Hence, *COSS'* and subsequently *COSS* are NP-Hard. In the *Knapsack* problem, we are given a space constraint B, and a set D of data items, where item d_k has space requirement $s(d_k)$ and benefit $b(d_k)$. We wish to select a subset D' of the items, so as to maximize the total benefit $\sum_{d \in D'} b(d_k)$, subject to the constraint $\sum_{d \in D'} s(d_k) \leq B$. Let *COSS'* be an instance of *COSS*, where each and every object u_k in U has unit benefit (i.e., $b(u_k) = 1$), and requires one and only one component v_k from V with space requirement $s(v_k)$. Associate each component v_k with $b(d_k)$ objects from U, and let $s(v_k) = s(d_k)$. Then, the mapping of an item d_k into a component v_k concludes the reduction. □

Note that in the special case where all the queries have the same benefit value then the problem is one of trying to satisfy the largest number of objects possible given the space constraint.

4.3.1 On the Applicability of the COSS Problem

Observe that the *COSS* problem is orthogonal to the problem of how to use the information from the marginals in order to reconstruct the corresponding queries. The query reconstruction problem takes as input a specific set of marginals and estimates the unknown values that produced them. In contrast, the *COSS* problem is one of selecting which of all the available marginals to materialize in a space constrained environment.

Note that the set of marginals which serves as input to the *COSS* problem, can be carefully selected to be representative of the query workload. The state of the art commercial database management systems offer efficient tools that, based on some query workload, can automatically generate the list of marginals that are most useful in answering the specified queries [ZCL+00] [ACN00] [LL02]. Then, these marginals are candidates for materialization. The assumption during this process is that the past workload is indicative of the future queries that will be posed on the system. Furthermore, queries that are similar to each other can be indentified [CGN02], and the proposed solution can be generalized to suit classes of queries, rather than the specific queries present in the input workload. In this way, the selected marginals will be useful for answering future queries that are different from, yet similar to, the queries in the past workload.

The *COSS* problem is also applicable to the domain of *pervasive computing* that has recently gained attention in the research community [CFZ01b]. The premise of pervasive computing is that all the relevant information (at each point in time and space) should be accessible to any mobile user. The users specify what information is relevant by using a profile language. This language allows each user to define a number of objects that she wants to be satisfied by the data components in the cache of the computer. Each object is associated with a benefit value, and needs a set of the data components in order to be satisfied. Furthermore, each data component may help satisfy more than one object.

One of the problems in the context of pervasive computing, as considered in *profile-driven data management* [CFZ01a], is what data to send to a mobile computer. The choice of the data to download is crucial, because it determines which of the objects in the profile can be satisfied locally, and consequently the total benefit that can be achieved.

Note that the same problem arises even when the amount of data is not large, but when the actual available storage space is limited. Characteristic examples of this scenario are handheld devices, which have small storage capacity, and for which space allocation should be done with care. Mobile users in general face an analogous problem in terms of bandwidth. When the available bandwidth or connection time resources are limited, we have to choose carefully which bytes to transmit over the link.

We will now describe a simple example of the above scenario. Consider a user visiting a new city. The profile of such a user could contain the following three statements (each

statement specifies an object and the components we need to satisfy the object).

1. Find location, satisfied by map m.

2. Find rental car, satisfied by rental car agency list c, and map m.

3. Find restaurant, satisfied by restaurant list r, and map m.

In this example objects 2 and 3 cannot be satisfied if only one of the two required data components are present in the mobile computer, and therefore the benefit gained is zero. Also observe that the map component m is useful in satisfying all 3 objects.

More formally, the problem in this environment can be defined as follows. Let V denote a finite set of data components that are candidates for transmission to a mobile computer, and $s(v), v \in V$ be the space requirement of data component v. Let the profile of the mobile user specify a set of objects U, where each object $u \in U$ has a benefit $b(u)$, and is associated with a subset of components from V, i.e., the data components which are necessary in order to satisfy the object and get the benefit. Assume the space constraint is denoted by W, the set of selected data components is $V_{sel} \subset V$, and the set of objects that can be satisfied using V_{sel} is $U_{sel} \subset U$. Then we have the *Data Recharging* optimization problem.

Problem 4.2 *Maximize* $\sum_{u \in U_{sel}} b(u)$, *given* $\sum_{v \in V_{sel}} s(v) \leq W$.

We observe that the formulation of the above problem is the same as in the *COSS* problem (Problem 4.1). Therefore, all the algorithms described in this chapter can be applied for the solution of the profile-driven data management problem as well.

4.4 Algorithms for the *COSS* Problem

In the following subsections, we propose several different algorithms for the solution of the *COSS* problem. We start by discussing the exhaustive algorithm, and then present efficient heuristics that can scale up to realistic sizes of the problem. We also explore the applicability of a randomized algorithm for the problem at hand.

We demonstrate the efficiency of the above algorithms, and compare their performance with a set of experiments in Section 4.5.

4.4.1 Exhaustive Enumeration

There are two reasons we include an exhaustive algorithm in our discussion. First, it will demonstrate the dramatic difference in execution time between the exhaustive algorithm, which always identifies the optimal solution, and the heuristics. There are $O(2^{|U|})$ possible solutions to be considered, where $|U|$ is the number of objects. When the number of objects is more than one or two dozen the above approach becomes prohibitively expensive. (An instance of the problem with just 20 objects was running for more than 10 days on our machine.) For the applications we have in mind, the number of objects is typically in the range of several hundreds.

Second, it will serve as a basis for comparison of the quality of the results with the heuristic approaches. Unfortunately, because of the complexity of the problem, this comparison will only be feasible for very small sizes of the problem.

In order to find the optimal solution, it is not necessary to enumerate and examine every single possible answer in the solution space explicitly. It is easy to determine for many of the answers that they are sub-optimal without examining them. Solutions with sufficiently low cardinality (such that they are much lower than the space constraint) are bound to be sub-optimal, because we can always add new objects to the solution, thus increasing the benefit. Similarly, for answers that have already exceeded the space constraint we can safely prune all the solutions with larger space requirements. Finally, we can also prune the solutions that we know will never improve on the current best solution. These are the partial solutions in which, even if we fill up all the remaining space with the currently smallest objects, while assuming that these objects carry the current maximum benefit, the total benefit will still be smaller than the highest benefit we have seen so far. We will call the straight forward exhaustive algorithm *Naive*, and the one that prunes the search space *NaivePrune*.

Note that dynamic programming cannot help find the optimal solution in reasonable time. Dynamic programming has been applied for the solution of the *Knapsack* problem, some instances of which can be solved in pseudo-polynomial time [GJ79]. However, this technique cannot be applied in our case. The intuition behind this observation is the following. Assume that at some point we want to compute the optimal solution for a subset of the objects (or equivalently components) and for a fraction of the total space allowed. It turns

out that there may be more than one optimal solution at this point. Given that the current solution may affect the later choices of the algorithm, we need to keep track of all of them, which leads to exponential time (or space). Thus, the dynamic programming algorithm will not terminate in pseudo-polynomial time (remember that dynamic programming can solve the *Knapsack* problem in time polynomial in the size of the input).

What makes the *COSS* problem difficult is the fact that, in order to satisfy an object, we need *all* the corresponding components. Materializing a subset of the components for some object does not give us part of the object's benefit.

4.4.2 Solutions Based on Bond Energy

We now present efficient heuristic algorithms based on the *Bond Energy Algorithm.* [MSW72], A high-level description of the algorithm we propose is shown in Figure 4.2.

The algorithm starts by computing a measure of interrelation between each pair of components $\{v_i, v_j\} \subset V$. The interrelation measures for all possible pairs of components are captured in the square matrix A (line 5 of the algorithm):

$$A[i,j] = \sum_{u_k | u_k \in U, e_{ki} \in E, e_{kj} \in E} b(u_k),$$

where e_{ki} is an edge between the k-th object and the i-th component and e_{kj} is an edge between the k-th object and the j-th component. This measure is a function of the benefit of the objects that require both components, v_i and v_j. The larger the number of objects and their benefits, the stronger the connection between the pair of components is.

Then, the procedure $MakeBlockDiag()$ permutes the columns of A (or equivalently the rows since A is symmetric by definition), and transforms it into a semiblock diagonal form. In this form, large interrelation values tend to be surrounded by large values as well, and are distinguished from smaller values that form separate groups. This transformation is expressed by the formula

$$\max \left(\sum_{k=1}^{|V|} \sum_{l=1}^{|V|} A[k,l](A[k,l-1] + A[k,l+1] + A[k-1,l] + A[k+1,l]) \right),$$

where $A[k,0] = A[0,l] = A[|V|+1,l] = A[k,|V|+1] = 0$. The above formula is the mathematical representation of the bond energy [MSW72]. We are seeking the maximum

Objective: Compute a solution (may be sub-optimal) for the *COSS* problem.
Input: A set U of objects, a set V of components, a space constraint W, and the bipartite graph $G(U, V, E)$, where $e_{ij} \in E$ denotes that component v_j is used by object u_i.
Output: A set of components $S \subset V$ to materialize.

```
1 procedure SolveCOSS() {
2       let square matrix A[·] = 0;
3       let S = ∅;                  /* selected components S ⊂ V */
4       for i,j=1 to |V|
5           A[ij] =interrelation measure between components i and j;
6       A[·] = MakeBlockDiag(A[·]);
7       S = Split(A[·]);
8       return(S);
9 }

10 procedure MakeBlockDiag(A[·]) {
11      let matrix A'[·] be empty;
12      for i=1 to |V| − 1 {
13          select one of the remaining columns in A[·];
14          let f_max = 0;              /* maximum increase in bond energy seen so far */
15          for j=1 to i+1 {
16              place new column in A'[·] in position j;
17              f_max = max(bond energy increase after placing new column in position j, f_max);
18          }
19          keep the A'[·] corresponding to f_max;
20      }
21      return(A'[·]);
22 }

23 procedure Split(A[·]) {
24      let f_max = 0;              /* total benefit of best solution seen so far */
25      for i=1 to |V| {
26          make first column of A[·] last, and first row last;
27          let j:=1;
28          let S = ∅;              /* selected components S ⊂ V */
29          while (j< |V| ∧ components in S satisfy the space constraint) {
30              S = S ∪ {component v corresponding to j-th column of A[·]};
31              f_max = max(total benefit of objects satisfied by components in S, f_max);
32          }
33      }
34      return(S corresponding to f_max);
35 }
```

Figure 4.2: The *BondEn* algorithm for the solution of the *COSS* problem.

value of this expression over all possible arrangements of the columns of matrix A. In order to transform matrix A into its new form, we start with an empty matrix in which we add one new column at a time (Figure 4.2, procedure $MakeBlockDiag()$). The new column is added in that position of the new matrix that results in the largest contribution to the increase of the overall bond energy. Using this stepwise technique we can focus our attention on the amount of *increase* of the bond energy after inserting a new column, since selecting at each step the column with the highest increase leads to the maximum value for the overall bond energy. Therefore, it suffices at each step to compute the formula

$$\arg\max_{1\leq k\leq |V|} \left(\sum_{l=1}^{|V|} A[l,k-1]A[l,k] + \sum_{l=1}^{|V|} A[l,k]A[l,k+1] - \sum_{l=1}^{|V|} A[l,k-1]A[l,k+1] \right),$$

when the new column is being placed at position k of the matrix. The above procedure is very fast, avoiding examination of the entire exponential search space, and is still able to find a near optimal form for A in almost all cases [MSW72].

In the final step, the algorithm splits the set of components V into H and $V - H$, where H is the set of components selected for materialization. Essentially, H is the set of components that can be used to satisfy the largest fraction of high-benefit objects while restricting the amount of available space. Because of the previous step of the algorithm, we do not have to examine arbitrary sets of components when constructing H, but rather neighboring components, i.e., only consecutive columns of A. Just a single split point in A determines H and $V - H$. The loop in line 28 makes sure that we will not miss good solutions for H even if H was originally situated in the centre of A. We will refer to this algorithm as *BondEn (Bond Energy)*. Its time complexity is $O(|V|^3)$.

Note that in the above approach, in order to transform matrix A into a semiblock diagonal form, we add one column at a time in the matrix position that results in the largest increase to the overall bond energy. However, the selection of the new column to add is arbitrary. Instead, we can enhance the technique by introducing a greedy column selection approach. That is, by choosing at every step among all the available columns (i.e., the ones not already placed) the one that leads to the highest bond energy. We will call this version of the algorithm *BondEnGr (Bond Energy Greedy)*. Obviously, this enhancement comes at the cost of increased computation time complexity, which now becomes $O(|V|^4)$. We also experimented with two other variations of the algorithm, where we make sure

that during the split step (i.e., the last phase of the algorithm), we do not include in H any components required by objects that are not satisfied. We call the above variations *BondEn-SpAll* and *BondEnGr-SpAll*, which are the extentions of *BondEn* and *BondEnGr* respectively.

4.4.3 Solutions Based on Greedy Algorithms

The greedy algorithms provide very fast alternatives for solving the *COSS* problem (albeit, they may provide a sub-optimal solution), and are particularly appealing for very large instances of the problem. The skeleton of such algorithms is depicted in Figure 4.3. They

Objective: Compute a solution (may be sub-optimal) for the *COSS* problem.
Input: A set U of objects, a set V of components, a space constraint W, and the bipartite graph $G(U, V, E)$, where $e_{ij} \in E$ denotes that component v_j is used by object u_i.
Output: A set of components $S \subset V$ to materialize.

```
1 procedure SolveCOSS() {
2       let O = ∅;                  /* selected objects O ⊂ U */
3       let S = ∅;                  /* selected components S ⊂ V */
4       while (|S| < |V| ∧ components in S satisfy the space constraint){
5           among all the objects {u|u ∈ U ∧ u ∉ O} select u_i that satisfies the greedy condition;
6           let O = O ∪ {u_i};
7           let S = S ∪ {v_j|e_ij ∈ E ∧ v_j ∉ S};
8       }
9       return(S);
10 }
```

Figure 4.3: The greedy algorithm for the solution of the *COSS* problem.

start with an empty solution set, and at each step they add to the solution set those components that satisfy the greedy condition. Before discussing the alternatives, we introduce some new notation that will be necessary for the mathematical formulation. Let S^k be the set of components that the algorithm has selected during the past k iterations, and O^k be the set of objects that are satisfied given the components in S^k. We are interested in deciding how to update those sets during iteration $k+1$. Let $CO^k = U - O^k$ be the set of candidate objects for selection during iteration $k+1$. Let $C_l = \{v_j | e_{lj} \in E \wedge v_j \notin S^k\}$ be the set of components, which are not in S^k, required for object $u_l \in CO^k$. The additional amount of space required by these components is given by the formula $f(u_l) = \sum_{v_j \in C_l} s(v_j)$.

As before, we assume that we know the graph $G(U, V, E)$, where U is the set of objects, V is the set of components, and E is the set of edges indicating which components are needed in order to satisfy each object. The greedy step can take any of the following four forms.

1. Accept to the solution set those components that will satisfy at least one new object, and that require the least amount of space. More formally, for each object $u_i \in CO^k$, calculate the additional space required for storing the corresponding components, given by $f(u_i)$. Let $u_l = \arg\min_{u_i} f(u_i)$ be the object that can be satisfied using the least additional space. Then, $O^{k+1} = O^k \cup u_l$, and $S^{k+1} = S^k \cup C_l$.

   This approach attempts to satisfy as many objects as possible, by selecting at each step the object with the least additional space requirements. The intuition is that if there are many objects answered then the total benefit of those objects will be high. However, this may not be the case if all the satisfied objects happen to have low benefit values. The time complexity of the above algorithm is $O(|U|^2)$, and we will refer to it as *GrSp (Greedy Space)*.

2. Accept to the solution set those components that will satisfy the new object with the highest associated benefit. More formally, let $u_l = \arg\max_{u_i \in CO^k} b(u_i)$ be the object with the highest benefit among all the candidate objects. Then, $O^{k+1} = O^k \cup u_l$, and $S^{k+1} = S^k \cup C_l$.

   The goal of this alternative is to satisfy as many of the high-benefit objects as possible, by choosing them according to their benefit values. This approach is likely to fail in the case where the required components have exceptionally high space requirements. We will refer to this algorithm as *GrBen (Greedy Benefit)*. This is the cheapest of the alternatives with time complexity $O(|U|^2)$.

3. Accept to the solution set those components that will satisfy a new object, and the ratio of the object benefit over the space required by the additional selected components is minimal. More formally, for each object $u_i \in CO^k$, calculate the additional space required for storing the corresponding components, given by $f(u_i)$. Let $u_l = \arg\max_{u_i} \frac{b(u_i)}{f(u_i)}$ be the object that has the highest benefit per unit of additional space required by its components. Then, $O^{k+1} = O^k \cup u_l$, and $S^{k+1} = S^k \cup C_l$.

   This variation of the greedy algorithm attempts to correct for the extreme cases we

identified as weaknesses in the previous two alternatives. The choice is now based on the object benefit per unit of additional required space, which prevents the algorithm from pursuing extreme solutions. We will refer to this algorithm as *GrBenSp (Greedy Benefit per unit Space)*. Its time complexity is $O(|U|^2)$.

4. Accept in the solution at each iteration an individual component. Select the component that fits in the remaining space and yields the maximum B/s ratio, where B is the total benefit of all the objects that are now satisfied because of the selection of the new component, and s is the space requirements of the new component. If the selection of no component causes any new objects to be satisfied then the algorithm picks the component with the smallest space requirement.

   This algorithm explores the applicability of choosing components instead of objects. We do not expect it to perform well when most of the objects require more than one component in order to be satisfied. The time complexity of this algorithm is $O(|V|^2)$ and we will refer to it as *GrComp (Greedy Component)*.

4.4.4 Other Empirical Approaches

In what follows, we discuss the use of *simulated annealing*, a randomized algorithm, and *tabu search*, a metaheuristic technique, for solving the *COSS* problem. The above methods have been used extensively in the past for solving a variety of problems, including scheduling, routing, and graph optimization [JAMS89] [IW87] [GL97]. Such algorithms have the ability to avoid getting stuck in local minima, and, therefore, can explore a larger area in the solution space than greedy algorithms.

Simulated Annealing

Simulated annealing [KGV83] is a randomized hill climbing algorithm. In the following discussion we will refer to Figure 4.4, which presents the algorithm. We start with an initial solution (lines 2 and 3) that may either be arbitrarily chosen, or carefully selected according to some other algorithm. We choose to use a solution provided by one of the greedy algorithms we presented earlier as the initial solution. The goal is to find the global maximum of the objective function.

Lines 7-18 implement the hill climbing procedure. In simulated annealing, apart from

Objective: Compute a solution (may be sub-optimal) for the *COSS* problem.
Input: A set U of objects, a set V of components, a space constraint W, and the bipartite graph $G(U, V, E)$, where $e_{ij} \in E$ denotes that component v_j is used by object u_i.
Output: A set of components $S \subset V$ to materialize.

```
1 procedure SolveCOSS() {
2       let O = O_0;                    /* selected objects O ⊂ U */
3       let S = S_0;                    /* selected components S ⊂ V, corresponding to O */
4       let T = T_0;                    /* temperature of the system */
5       let delta = 0;
6       let f_max = 0;                  /* total benefit of best solution seen so far */
7       while (T > 1){
8           for i=1 to total number of iterations{
9               O_new =GetNewSimAnSolution(O);
10              let delta =(total benefit of O_new)− (total benefit of O);
11              if (delta ≥ 0)
12                  O = O_new;                  /* accept the new solution */
13              if (delta < 0) then with probability e^(−delta/T)
14                  O = O_new;                  /* accept the new solution */
15              f_max = max(total benefit of O, f_max);
16          }
17          T = αT;                     /* α < 1 */
18      }
19      S = marginals needed to answer queries in the O corresponding to f_max;
20      return(S);
21  }

22  procedure GetNewSimAnSolution(O) {
23      let S = marginals needed to answer the queries in O; /* S updated whenever O changes */
24      pick an object O_admit ∈ {U − O} at random, and insert it to O;
25      if (total space needed by S > W)
26          pick an object O_evict ∈ {O − O_admit} at random, and remove it from O;
27      if (total space needed by S > W)
28          penalize the total benefit of O proportionally to the benefit per space ratio of O_admit
                and the amount by which W is exceeded;
29      return(O);
30  }
```

Figure 4.4: The *SimAn* algorithm for the solution of the *COSS* problem.

uphill moves, downhill moves are also allowed under certain circumstances. More specifically, a downhill move is accepted with probability $e^{-(delta)/T}$ (line 13), where *delta* is the decrease in the objective function from the previous step, and T is a parameter simulating the temperature of the system. When T is high (in the beginning of the process) the probability of accepting downhill moves is high. Then, T is slowly decreased (line 17), and when the system freezes ($T < 1$) no further moves are considered, and the algorithm terminates. The temperature T is reduced according to the formula $T_{new} = \alpha T_{old}$, where α is a parameter controlling the rate of reduction of T. Note that this process involves two loops. The inner loop (lines 8-16) searches for solutions while T remains fixed. At each temperature, the simulation must proceed long enough for the system to reach a steady state or equilibrium. In our implementation, the equilibrium condition is satisfied when a constant number (depending on the problem size) of iterations have been executed.

The function *GetNewSimAnSolution()* (lines 22-28) determines what the proposed solution for the next iteration of the algorithm is going to be. At this stage of the algorithm, the solution is expressed in terms of the selected objects, rather than the selected components. (This choice is simply for convenience, and has no effect in our reasoning.) The next solution is chosen at random among all the neighbours of the current solution. Two solutions are neighbours if we can derive one from the other by adding a single object to one of the solutions, possibly followed by a deletion of another object. This last step ensures that the current solution satisfies the space constraint (lines 25-26). Actually, in some cases we may have to remove more than one object in order to satisfy the space constraint. Nevertheless, the requirement that the current solution should satisfy the space constraint at every step is not strict. In some cases, we allow the new solution to violate the space constraint at the cost of a small penalty to the total benefit. Then, in subsequent iterations, we once again enforce the space constraint.

The complexity of the simulated annealing approach is determined by the number of iterations. The work done in each iteration is minimal, and we can safely consider it as constant. The number of iterations is controlled by the user, who sets the temperature parameter, and defines the way temperature is reduced, as well as the criteria for the equilibrium states. In our experiments, the initial temperature was set to four times the total benefit of the initial solution, the α parameter controlling the decrease of the temperature was set to 0.95, and the equilibrium condition involved a number of iterations equal to 1/3

of the total number of objects. Varying the above parameters did not have significant effects on the quality of solution found.

Tabu Search

Tabu search [GL97] is an algorithm for guiding known heuristics to avoid getting stuck in local optima. A structure called *tabu list* is used as auxiliary memory by the algorithm. The tabu list describes a set of moves that are not permitted. This way the algorithm can avoid revisiting solutions that were previously visited, and thus it is less probable that it will get stuck in local minima or cycles (i.e., examining the same solutions over and over again).

The outline of the tabu search algorithm for the *COSS* problem is shown in Figure 4.5. The algorithm starts with an initial solution (lines 2 and 3), which in our implementation is derived using the greedy heuristics we presented earlier. Then, the main loop (lines 6-11) iterates over the proposed solutions in order to pick the best one. The function *GetNewTabuSolution()* (lines 15-21) selects the solution (defined in terms of the selected objects) that will be examined during the next iteration of the algorithm. The new solution must be a neighbour of the current solution , and can be constructed in a variety of ways. We choose to use a greedy approach in order to select the new object to add to the solution. The last step of the function is to make sure that the proposed solution satisfies the space constraint (lines 18 and 19). However, this requirement is not strict. In some cases, we may allow a solution to exceed the space limitations, but its benefit value is penalized. Then, in subsequent iterations, we once again enforce the solution to be within the specified space constraint.

Note that the changes that lead to the new solution cannot involve any of the objects in the tabu list Q_{tabu}. In our implementation, the tabu list keeps track of the recently added or deleted objects. Nevertheless, this restriction can be overridden when the new solution is the best obtained so far. When the new solution is available, we update the tabu list (line 10), and proceed to the next iteration.

Similar to the simulated annealing method, the duration of tabu search is controlled by the user. When more time is allowed to tabu search to perform more iterations, more of the solution space is explored. An important difference in the case of tabu search is the selection of the next solution. It can be as simple as a random change in the current solution

Objective: Compute a solution (may be sub-optimal) for the *COSS* problem.
Input: A set U of objects, a set V of components, a space constraint W, and the bipartite graph $G(U, V, E)$, where $e_{ij} \in E$ denotes that component v_j is used by object u_i.
Output: A set of components $S \subset V$ to materialize.

```
1 procedure SolveCOSS() {
2       let O = O_0;                    /* selected objects O ⊂ U */
3       let S = S_0;                    /* selected components S ⊂ V, corresponding to O */
4       let f_max = 0;                  /* total benefit of best solution seen so far */
5       let O_tabu = ∅;                 /* set of tabu objects */
6       while (search not finished){
7           O_new =GetNewTabuSolution(O, O_tabu);
8           f_max = max(total benefit of O_new, f_max);
9           O = O_new;
10          update O_tabu based on the changes to O;
11      }
12      S = components needed to satisfy objects in the O corresponding to f_max;
13      return(S);
14 }

15 procedure GetNewTabuSolution(O, O_tabu) {
16      let S = components needed to satisfy objects in O; /* S updated whenever O changes */
17      pick an object O_admit ∈ {U − O − O_tabu} and insert it to O;
18      if (total space needed by S > W)
19          pick an object O_evict ∈ {O − O_admit − O_tabu} and remove it from O;
20      if (total space needed by S > W)
21          penalize the total benefit of O proportionally to the benefit per space ratio of O_admit
                and the amount by which W is exceeded;
22      return(O);
23 }
```

Figure 4.5: The *Tabu* algorithm for the solution of the *COSS* problem.

(like simulated annealing), or as involved as exhaustive enumeration. Therefore, there is a tradeoff between the number of iterations the algorithm will execute, and the complexity of each iteration (i.e., the reasoning power of the procedure for selecting the next solution). In our experiments, we use a greedy algorithm to select the new solution.

4.5 Experimental Evaluation

In the following sections, we present the experiments we used to evaluate the efficiency and behavior of the proposed algorithms.

4.5.1 Description of Experiments

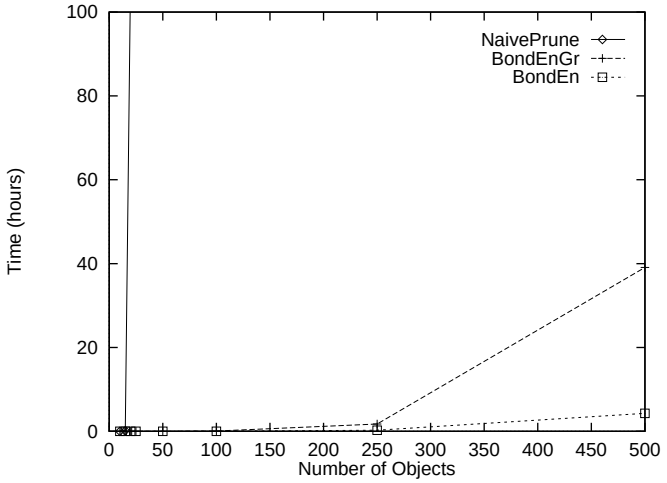
For the evaluation of the algorithms, we used synthetic datasets. The number of objects in the datasets we produced ranges from 10 to 1000. The number of components is in each case approximately twice the number of objects, and more than half of the components help satisfy multiple objects. In order to assign benefit values to objects and space requirements to components, we generated random numbers following uniform, Gaussian, and Zipfian distributions.

The benefit values drawn from the uniform distribution were between 10 and 100; the μ and σ parameters for the Gaussian distribution were 55 and 18 respectively; and the skew parameter for the Zipfian distribution was set to 1. The space requirements were between 10 and 1000 for the uniform distribution; the μ and σ parameters for the Gaussian distribution were 550 and 180 respectively; and the skew parameter for the Zipfian distribution was set to 1. Note that while the choice of the distribution is relevant in our experiments, the specific choice of the parameter values is not significant.

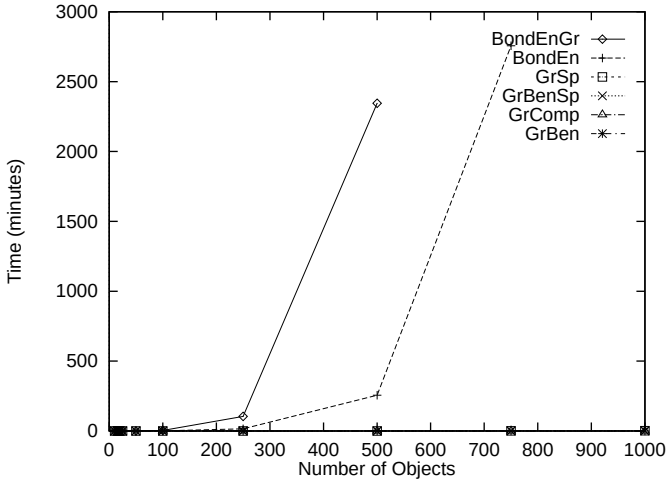
In the experiments, we measure the sum of the benefits of the objects that are satisfied given the components that are selected by the algorithms. We are also interested in the computation times of the proposed algorithms. Therefore, we report the time requirements of the proposed solutions as the number of objects increases. In all cases, we report the benefit of the solutions normalized by the total benefit of all the objects in the problem. Similarly, the space constraint is normalized by the total space requirements of all the components in the problem.

4.5.2 Scalability of the Algorithms

The first set of experiments examines the efficiency of the algorithms in terms of the time required to produce the solution. Figure 4.6 shows how the run-times of the algorithms changes as the number of objects increases. In Figure 4.6(a) we depict the tremendous



(a) NaivePrune and bond energy algorithms



(b) Bond energy and greedy algorithms

Figure 4.6: Scalability of the algorithms as a function of the number of objects.

difference in the computation time needed by the naive approach and the rest of the algorithms. Naive, which is represented by the near vertical line at the very left of the graph, is able to produce answers in a reasonable time-frame only for problems involving fewer than 20 objects. As shown in Figure 4.6(b) the bond energy algorithms scale more gracefully.

Nevertheless, when the problem size becomes large, i.e., more than 600 objects for *BondEn* and more than 400 objects for *BondEnGr*, these algorithms require more than 24 hours to produce a solution. Only the greedy algorithms are able to scale to thousands of objects in our examples. The time they required was under 2 minutes in all cases we considered, that is, in instances of the problem with up to 1000 objects.

4.5.3 Evaluating the Quality of the Solutions

In this set of experiments, we evaluate the quality of the solutions produced by the proposed algorithms. First, we compare the quality of the solutions produced by the bond energy family of algorithms. Figure 4.7 illustrates the benefits for the solutions produced by *BondEn*, *BondEnGr*, *BondEn-SpAll*, and *BondEnGr-SpAll* for various values of the space constraint. The differences among the algorithms are minimal in all cases that we considered. Observe

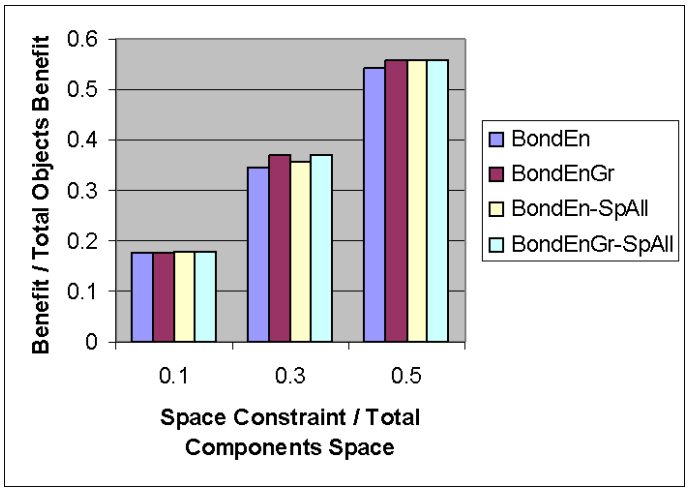


Figure 4.7: Comparison of the bond energy family of algorithms. Benefits and space requirements follow uniform distributions.

also that the greedy method of constructing the bond energy matrix (*BondEnGr*) in some cases improves the quality of the solutions. However, the more involved split procedure (represented by *BondEnGr-SpAll*) is only able to achieve a slightly better solution than the simpler approach (*BondEnGr*) in one of the experiments (i.e., when the space constraint is set to 0.1). Since the above algorithms perform without significant differences, for the rest of the experiments we only demonstrate *BondEn*, which is the fastest among them.

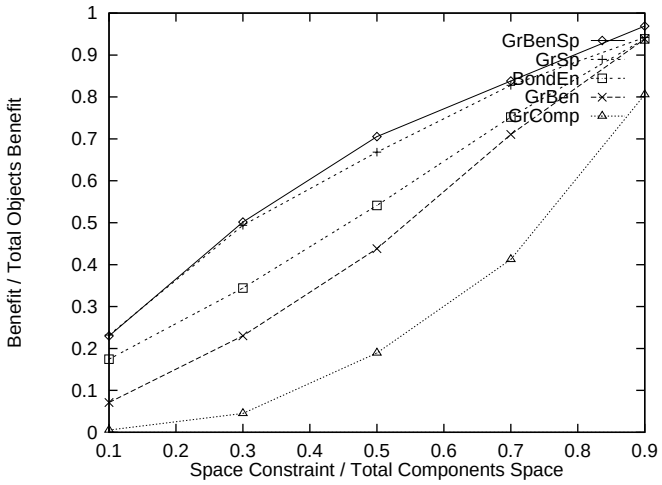
In the next set of experiments, we compare the quality of the solutions of the bond energy

and the greedy algorithms. Figure 4.8 illustrates the normalized total benefit of the solutions when we vary the space constraint. The three graphs correspond to the cases where the assigned object benefits and component space requirements follow uniform, Gaussian, and Zipfian distributions, respectively. In all three cases the graph G, which connects objects to components, remains the same. The best performance across all experiments is achieved by *GrBenSp*, closely followed by *GrSp*. The *BondEn* and *GrBen* algorithms perform in the middle range, while *GrComp* performs the worst. Figure 4.9 depicts the relative ordering of the algorithms in terms of the quality of the solution they achieve when we vary the graph G. As we move in the graph from left to right there is an increase in the average number of objects that are connected to each component (i.e., we increase the degrees of the component nodes in G). In all three cases the space constraint is 10% of the total space required by all the components. The relative performance of the algorithms is identical to the previous set of experiments, with *GrBenSp* being the best performer, and *GrComp* the worst.

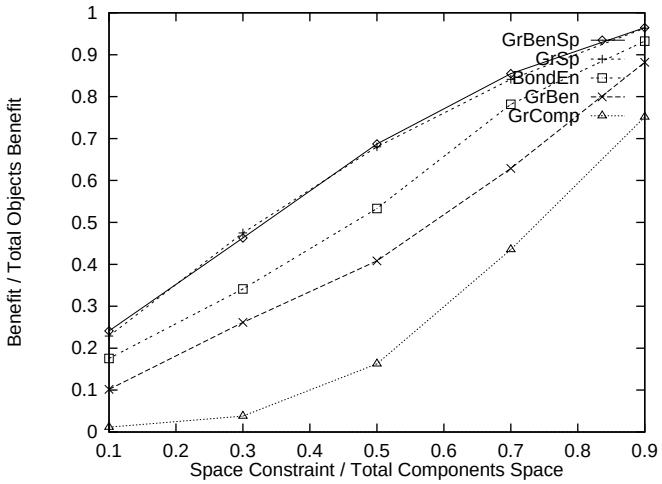
The poor performance of *GrComp* is explained by the nature of the algorithm, which builds the solution one component at a time, instead of sets of components like the rest of the algorithms. This choice restricts *GrComp*, and does not allow it to start accumulating benefit until all the components related to a specific object are brought in the solution. This explains the significantly slow rate at which the algorithm improves the solution at the lower left part of the graph in Figure 4.8(a).

We also conducted a series of experiments where we varied the number of objects. Figure 4.10 depicts the total benefit achieved by each algorithm. We report the results of running the algorithms on the same graph G, where the query benefits and the component space requirements were produced from uniform (Figure 4.10(a)), Gaussian (Figure 4.10(b)), and Zipfian (Figure 4.10(c)) distributions. The space constraint was in all cases set to half the total space required by all the components. These experiments show that the relative ordering in performance for all the algorithms we consider remains the same across various problem sizes.

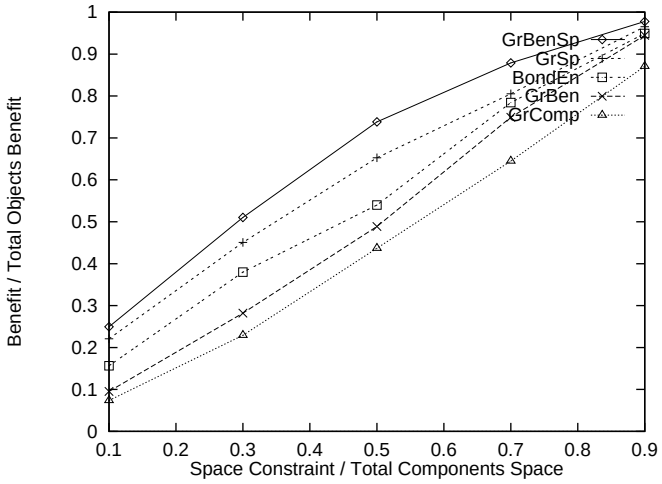
In Table 4.1 we report the results of the experiments with *SimAn* and *Tabu*. We use the solution provided by *GrBenSp* as the base solution, and report the improvement on this solution achieved by each one of the two algorithms. That is, for *SimAn* and *Tabu*, we



(a) Benefits and space requirements follow uniform distributions



(b) Benefits and space requirements follow Gaussian distributions



(c) Benefits and space requirements follow Zipfian distributions

Figure 4.8: Quality of the solution of the algorithms, when varying the space constraint.

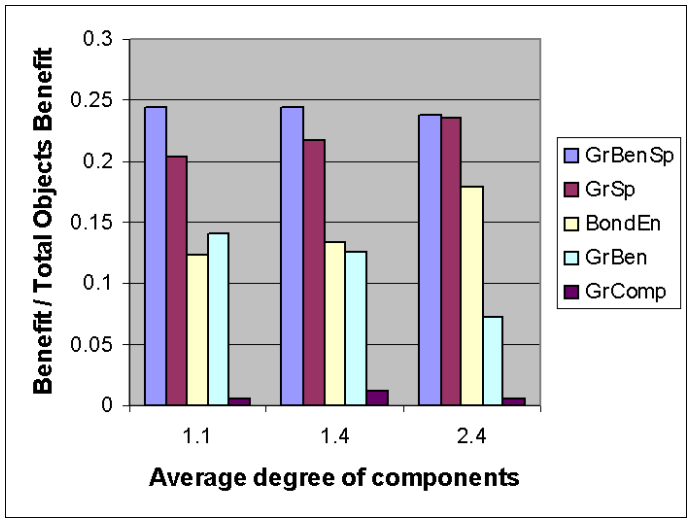


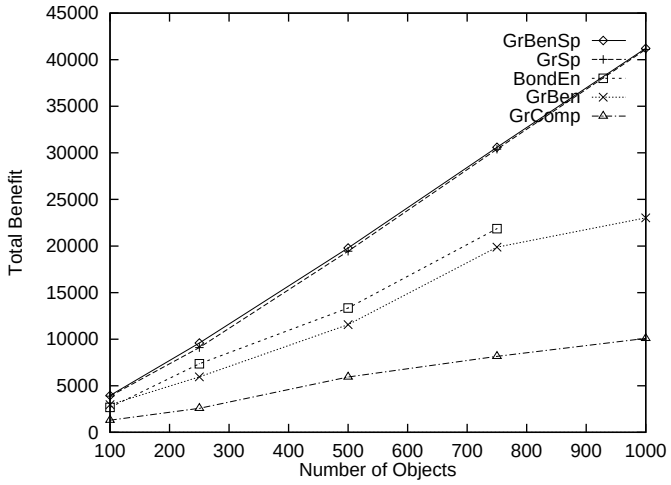
Figure 4.9: Quality of the solution of the algorithms, when varying the bipartite graph G.

report the ratio

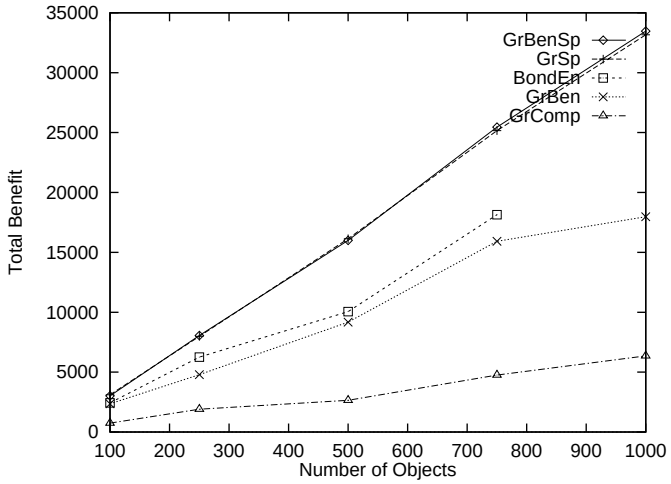
$$\frac{\text{(total benefit of } \textit{SimAn} \text{ or } \textit{Tabu} \text{ solution)} - \text{(total benefit of } \textit{GrBenSp} \text{ solution)}}{\text{total benefit of } \textit{GrBenSp} \text{ solution}}.$$

The initial solution for both *SimAn* and *Tabu* is provided by *GrBenSp*, and this is also the method used by *Tabu* to select solutions at each iteration. We experimented with varying the number of objects and the space constraint. In all the experiments we allowed the algorithms to run an equal amount of time to the time required by *GrBenSp* to produce the solution.

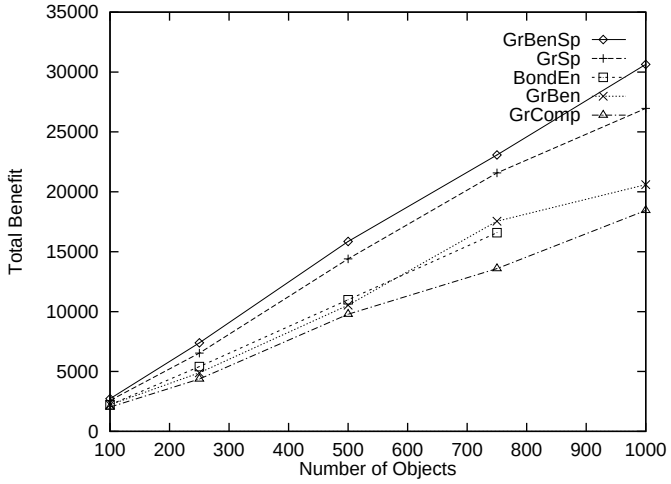
Unlike *SimAn*, *Tabu* was able to improve on the initial solution in all the cases tested. We believe that the reason *Tabu* outperforms *SimAn* is because the solution space is too large, and this makes it extremely difficult for *SimAn* to move to the correct direction. Remember that *SimAn* moves in the solution space by selecting at random one of the numerous solutions that are neighbors of the current solution. On the other hand, the *Tabu* algorithm directs its search in the solution space more effectively, because it employs a more structured way of taking steps at each iteration. We should note here that when we turned off the tabu list functionality the performance of the algorithm deteriorated. This indicates that *Tabu* is indeed making well-calculated moves, and that the tabu list enables the algorithm to avoid local minima and explore new areas in the solution space.



(a) Benefits and space requirements follow uniform distributions



(b) Benefits and space requirements follow Gaussian distributions



(c) Benefits and space requirements follow Zipfian distributions

Figure 4.10: Quality of the solution of the algorithms, when varying the number of objects.

		GrBenSp	SimAn	Tabu
objects	*constraint*			
100	*0.5*	1	**1.012**	**1.012**
250	*0.5*	1	1	**1.003**
500	*0.5*	1	**1.003**	**1.003**

(a) Varying the number of objects

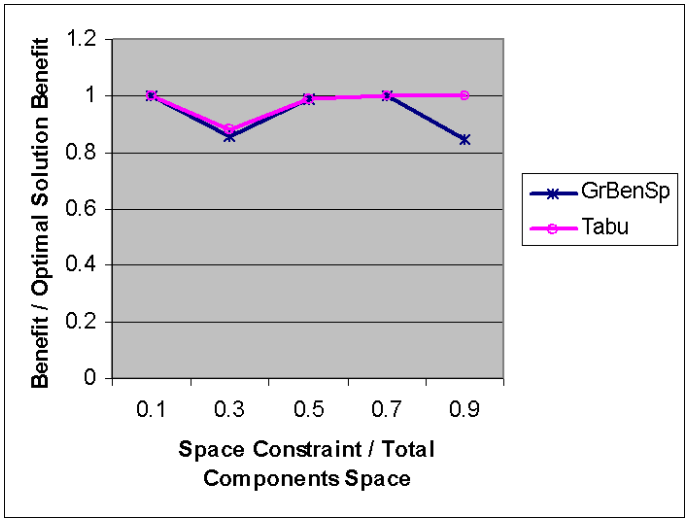
		GrBenSp	SimAn	Tabu
objects	*constraint*			
250	*0.1*	1	1	**1.05**
250	*0.3*	1	1	**1.002**
250	*0.5*	1	1	**1.003**

(b) Varying the space constraint

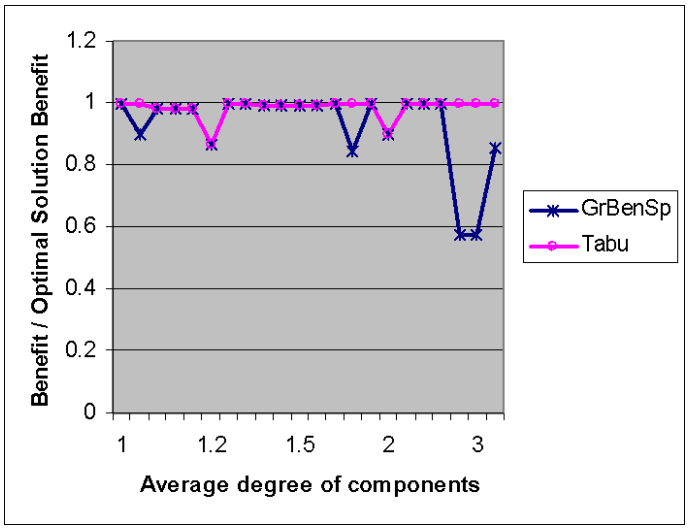
Table 4.1: Improvement in the *GrBenSp* solution by *SimAn* and *Tabu*.

An interesting observation is the fact that both *SimAn* and *Tabu* improve the solution only by a small amount (less than or equal to 5% for the cases we tested). A natural question then is how close to optimal is the solution provided by *GrBenSp* in the first place. Unfortunately, it is not easy to find the optimal solution for the experiments we presented, because the size of the problem is prohibitively large. Therefore, we conducted a series of experiments where the number of objects was set to 10, for which we could get the optimal solution. This allows us to compare the solutions provided by the algorithms to the optimal solution. The results of these experiments are depicted in Figure 4.11. The graphs show the benefit of the solutions produced by *GrBenSp* and *Tabu* normalized by the benefit of the optimal solution (which is always 1). In the graph shown in Figure 4.11(a) we vary the space constraint. In Figure 4.11(b) each point in the graph represents an experiment with a different graph G connecting objects to components. As we move in the graph from left to right there is an increase in the average number of objects that are connected to each component. The left-most point in the graph corresponds to the case where each component is connected to a single object, and the problem degenerates to the Knapsack problem.

Note that in many settings in both graphs *GrBenSp* finds a solution very close to optimal. Though, there are also cases where it achieves slightly more than half the benefit of the optimal. These experiments indicate that the *GrBenSp* algorithm is in many circumstances effective at finding near-optimal solutions. This explains the fact that *SimAn* and *Tabu*



(a) Varying the space constraint



(b) Varying the bipartite graph G

Figure 4.11: *GrBenSp* and *Tabu* compared to optimal.

could not find much better solutions than the greedy algorithm in our previous experiments (see Table 4.1). Nevertheless, the graphs show that the *Tabu* algorithm is able to improve upon *GrBenSp*, and find a very good solution in most of the cases where *GrBenSp* performs poorly.

4.6 Discussion

In the following paragraphs, we investigate in more detail some properties of the algorithms. These are properties relevant to the ability of the algorithms to provide high quality solutions under different circumstances. More specifically, we examine the behavior of the proposed solutions when the space constraint is not a strict one. That is, when a solution that uses a little bit more of space is acceptable. Furthermore, we try to quantify the quality of the solutions in some worst case scenarios, when compared to the optimal solution. This analysis indicates how poor the performance of the algorithms can become under certain conditions. Finally, we conclude by summarizing the benefits and drawbacks of each approach, and by providing a practical guide for selecting among the proposed solutions.

4.6.1 Soft Space Constraints

So far we have made the assumption that the space constraint is a fixed number given in the input of the *COSS* problem, and the algorithm that provides the solution is not supposed to violate this constraint. However, it is not always the case that a strict space constraint is what we are after. Often, the space constraint is merely an indication or approximation of the amount of space that is available. In such cases, we may allow the solutions produced by the algorithms to exceed the space constraint by a small amount. Since we cannot easily quantify what a suitable small amount would be in each case, it is left to the user to examine the solutions, and decide whether the increase in benefit justifies the larger space requirements.

We now describe how we can change the algorithms in order to operate in this environment. The goal is to alter the algorithms so that they incrementally produce solutions which occupy more space and, naturally, have larger total benefit.

Bond Energy: The bond energy algorithms cannot readily provide the new solution, because they have to rerun the *Split* procedure (see Figure 4.2), which determines

which components should be in the solution. Running the *Split* procedure each time we allow more space for the solution is an expensive operation, requiring time $O(|V|^2)$. Each new solution is not necessarily a superset of the previous one, since we select the components from scratch.

Greedy: It suffices to change the condition in line 4 (see Figure 4.3), which checks whether the solution satisfies the space constraint. Once we remove this condition, the greedy algorithms can produce with linear complexity at each iteration a new solution with increased space requirements. At each step we are only adding new objects or components, thus, we get a solution that is a superset of the solution of the previous iteration.

Simulated Annealing: In this case we only need to remove the conditions in lines 25 and 27 (see Figure 4.4) that check if the current solution requires space more than the space constraint. Since the process of forming a solution involves a randomization step, there is no guarantee that the new solution will be a superset of the old one.

Tabu Search: Similar to simulated annealing, the only change is to remove the conditions of lines 18 and 20 (see Figure 4.5). Once again, the solutions produced by the algorithm are not guaranteed to be supersets of the previous solutions.

In the following section, we focus on the greedy algorithms. More specifically, we examine the performance of these algorithms (in terms of the quality of solutions) in relation to the performance of the greedy solution for the *Knapsack* problem.

4.6.2 On the Optimality of the Greedy Solutions

As we have already discussed, the *COSS* problem degenerates to the *Knapsack* problem when each component satisfies at most one object. In this case, we can consider the set of components required by each object as a single super-component, and attach to it a space requirement equal to the sum of the space requirements of the corresponding components.

We can also show that the following theorem holds.

Theorem 4.2 *Consider the* Knapsack *problem with n objects. Let the space constraint W be equal to the space needed by the i objects,* $1 \leq i \leq n$*, with the highest benefit per space*

ratio. In this case the greedy algorithm that chooses items based on their benefit per space ratio finds the optimal solution.

Proof: Figure 4.12 illustrates a linear ordering of the n objects according to their benefit per space values. We name u_1 the object that has the highest benefit per space ratio,

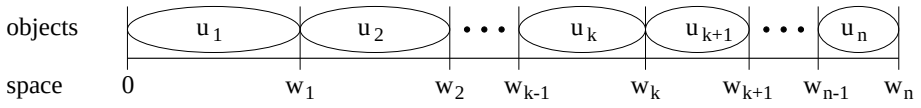


Figure 4.12: A linear ordering of n objects for the *Knapsack* problem.

and u_n the object that has the lowest. This linear ordering represents the order in which the greedy algorithm will select objects to insert in the solution. Assume that we set the space constraint to $W = w_{k+1}$, which is the total space required by the first $k+1$ objects. Then we solve the *Knapsack* problem using this space constraint. We know that the greedy algorithm that chooses items based on their benefit per space ratio is optimal (in benefit) for the *fractional Knapsack* problem [CLRS01]. Observe though, that in our case the space constraint is carefully set so that all the selected objects will fit exactly in the available space, and none will need to be divided. (Remember that the greedy algorithm will select the objects from left to right, in the order they appear in Figure 4.12.) Note that the same is true for all the choices of $W = w_i$, where $1 \leq i \leq n$. Therefore, the greedy algorithm that chooses items based on their benefit per space ratio is also optimal for the *Knapsack* problem when the space constraint is set to w_i, where $1 \leq i \leq n$. □

This theorem is interesting in our context. It says that, when we can relax the space constraint, we can solve the *Knapsack* problem using the greedy benefit per space algorithm, stop after selecting any number of items, and we are guaranteed that the solution we have is optimal for the amount of space used. Given the above analysis, a question that arises naturally is whether this optimality property carries over to the *COSS* problem and the corresponding algorithm *GrBenSp*.

Unfortunately, the answer is negative. The reason is that unlike *Knapsack*, in the *COSS* problem each component may help satisfy more than one object. The result of these interdependencies is that a solution that is optimal for some space constraint W_1 is not necessarily a subset of the optimal solution for a space constraint $W_2 > W_1$. Consequently, there is no linear ordering of the objects in the sense depicted in Figure 4.12. Hence,

the optimality property of the greedy algorithm does not hold for the *COSS* problem. Theorem 4.3 formally states this observation.

Theorem 4.3 *The* COSS *problem does not have the optimality property. That is, the optimal solution for some space constraint W_1 is not necessarily a subset of the optimal solution for a space constraint $W_2 > W_1$.*

Proof: We prove this theorem using a counter-example. Consider the instance of the *COSS* problem depicted in Figure 4.13. We indicate the benefit of each object and the

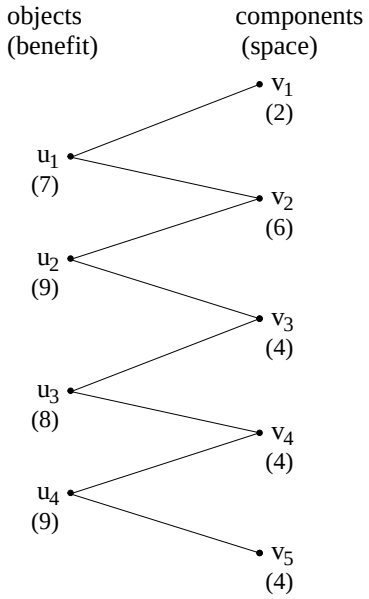


Figure 4.13: Counter-example that demonstrates the fact that for the *COSS* problem, the optimal solution for one space constraint value is not necessarily a subset of the optimal solution for larger space constraint values. The numbers in parentheses indicate the benefit and required space of the objects and components respectively.

space requirements for each of the components by the numbers in parentheses below the names of the objects and components. Assume that the space constraint is $W_1 = 12$ units. Then, the optimal solution for this particular instance of the *COSS* problem is the set of objects $O_1 = \{u_3, u_4\}$ with total benefit 17 units. Now let the space constraint be $W_2 = 16$ units. In this case, the optimal solution is composed of the set of objects $O_2 = \{u_1, u_2, u_3\}$ with total benefit 24 units. We observe that the optimal solution O_2 is *not* a superset of O_1, even though the space constraint W_2 is greater than W_1. □

4.6.3 Analysis of the Greedy Algorithms

In this section, we present theoretical results on the behaviour of the greedy algorithms when compared to optimal. These results are interesting, because they indicate what the worst performance of the algorithms might be, and they can help us choose among those algorithms.

In the following figures, we illustrate examples where the algorithms perform poorly in terms of quality of the solution. The benefit of each object and the space requirements for each of the components are indicated by the numbers in parentheses below the names of the objects and components. In all cases b represents a measure of the objects' benefit, while the space constraint is set to W units. We assume that b and W take values from the domain of positive integers $\mathcal{Z}^+$.

Theorem 4.4 *In the worst case, the* GrComp *algorithm provides a solution that is at least* $(\frac{W}{2} - 1)b$ *times worse than the optimal solution, for any* $b > 0$ *and* $W > 3$.

Proof: In the instance of the *COSS* problem depicted in Figure 4.14, *GrComp* will first select component v_1, because the selection of no component will yield any benefit, and v_1 has the minimum space requirements among all the components. Subsequently, it will select component v_2 in order to get the benefit from satisfying object u_0, for a total benefit of 1. After that there are no more objects that can be satisfied, since the total space required by the set of the selected components, i.e., $\{v_1, v_2\}$, is equal to the space constraint W. On the contrary, the optimal algorithm will select the components $\{v_3, \ldots, v_{W/2-1}\}$ ($W/2$ components in total), for a total benefit of $(\frac{W}{2} - 1)b$. Therefore, *GrComp* can be $(\frac{W}{2} - 1)b$ times worse than the optimal solution. □

Theorem 4.5 *In the worst case, the* GrBen *algorithm provides a solution that is at least* $(W - 1)(1 - \frac{1}{b})$ *times worse than the optimal solution, for any* $b > 0$ *and* $W > 2$.

Proof: In Figure 4.15, we give an example for *GrBen*. In this case the algorithm will accept object u_0 in the solution, because it has the highest benefit value among all the objects. The space required by u_0, or equivalently $\{v_1, v_2\}$, is W. Then the algorithm terminates, since it will have used up all the available space. The benefit for this solution is b. The optimal solution includes the objects $\{u_1, \ldots, u_{W-1}\}$, for a total benefit of $(W - 1)(b - 1)$.

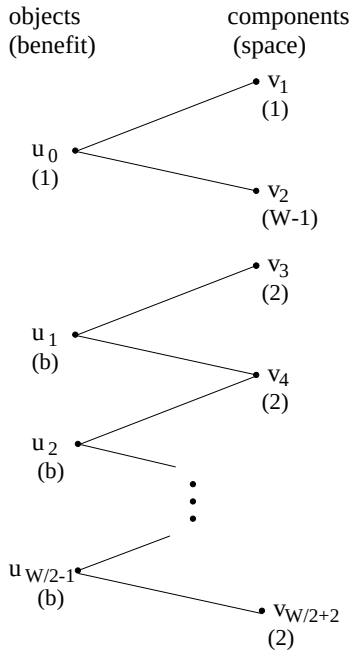


Figure 4.14: Example scenario for *GrComp*. The numbers in parentheses indicate the benefit and required space of the objects and components respectively.

This means that the solution of *GrBen* can be $(W-1)(1-\frac{1}{b})$ times worse than the optimal solution. □

Theorem 4.6 *In the worst case, the* GrSp *algorithm provides a solution that is at least* $1.33b$ *times worse than the optimal solution, for any* $b > 0$ *and in the limit as* $W \to \infty$.

Proof: For the *GrSp* algorithm consider the scenario depicted in Figure 4.16. The algorithm will select the objects $\{u_1, \ldots, u_{W/2}\}$. The components needed to satisfy these objects have the lowest space requirements, i.e., they occupy space of 2 units per object. In contrast, all the rest of the objects require space of 3 units each. The selected objects are $W/2$ altogether, they yield a total benefit of $W/2$, and the corresponding components occupy all the available space W. The optimal solution in this case is comprised of the objects $\{u_{W/2+1}, \ldots, u_{W/2+2W/3-1}\}$. The number of the selected objects is $(\frac{2W}{3}-1)$, and their total benefit is $(\frac{2W}{3}-1)b$. Thus, *GrSp* can be $\frac{(2W/3-1)}{W/2}b$ times worse than optimal. For sufficiently large values of W the above quantity is approximated by $1.33b$. □

Theorem 4.7 *In the worst case, the* GrBenSp *algorithm provides a solution that is at least* 1.77 *times worse than the optimal solution, in the limit as* $b \to \infty$.

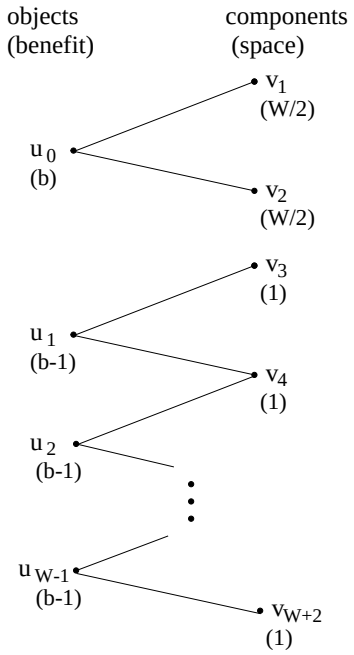


Figure 4.15: Example scenario for *GrBen*. The numbers in parentheses indicate the benefit and required space of the objects and components respectively.

Proof: Consider the example illustrated in Figure 4.17, for which we assume that the space constraint is $W = 2b$. During the first iteration the *GrBenSp* algorithm will select object u_3, because it has the largest ratio of benefit per space required by the corresponding components, which are $\{v_3, v_4\}$. This ratio is 1 for u_3, and less than 1 for all the other objects.

In the next iteration the object u_4 is selected. Note that in order to satisfy u_4 only the component v_5 is needed, since v_4 was selected in the previous iteration. The object u_4 is selected because its benefit per space ratio, $\frac{5b+2}{3b}$, is higher than the one of u_2, which requires component v_2, and has a ratio of $\frac{5b-4}{3b}$. The other two choices have lower ratios, as well. The object u_1, which requires components $\{v_1, v_2\}$, has a ratio of $\frac{b-1}{b}$, and object u_5, which requires components $\{v_5, v_6\}$, has a ratio of $\frac{1}{b}$.

In the third iteration the algorithm selects object u_5, which now only needs component v_6, since v_5 is already selected. Observe that the benefit per space ratio of this choice ($\frac{2}{b}$) is not better than the ration of u_1 ($\frac{b-1}{b}$). However, it is the only viable choice at this point, because the remaining available space is just $b/2$.

In summary, the *GrBenSp* algorithm selects the set of objects $\{u_3, u_4, u_5\}$, and the

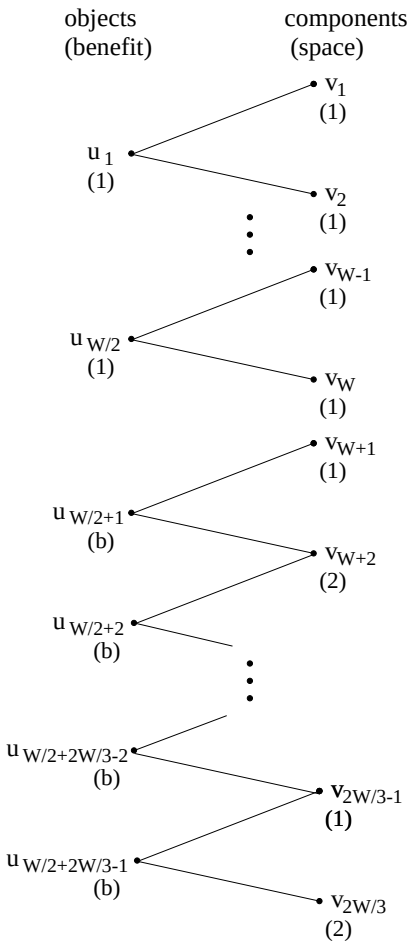


Figure 4.16: Example scenario for *GrSp*. The numbers in parentheses indicate the benefit and required space of the objects and components respectively.

corresponding components $\{v_3, v_4, v_5, v_6\}$, which occupy space $2b = W$. The total benefit of this solution is $\frac{11b+8}{6}$.

The optimal solution for this example consists of the set of objects $\{u_1, u_2, u_3\}$. The required components occupy all the available space W, and the total benefit of the solution is $\frac{13b-8}{4}$.

Therefore, the solution of *GrBenSp* can be $\frac{39b-24}{22b+16}$ times worse than optimal. For sufficiently large values of b the above quantity is approximated by 1.77. □

4.6.4 Choosing Among the Alternatives

The experiments demonstrate that the bond energy algorithms perform worse than some of the greedy approaches. At a first glance this is a rather surprising result, given that the

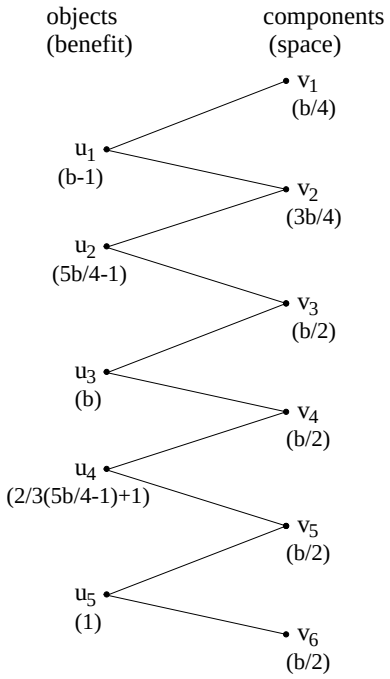


Figure 4.17: Example scenario for *GrBenSp*. The numbers in parentheses indicate the benefit and required space of the objects and components respectively.

BondEn algorithms have higher complexity and seem to make choices with greater care. However, we observe that all the decisions that *BondEn* makes are based on just *pairs* of components, despite the fact that in many cases it is a larger number of components that are interrelated. This prevents *BondEn* from capturing the true, more complex, associations inherent in the problem, narrows the information upon which the algorithm operates, and leads to poor decisions. Thus, *BondEn* should not be the algorithm of choice, especially since it does not scale as well as the greedy algorithms.

The greedy approaches are better than the bond energy algorithms for another reason as well. They are able to incrementally compute a new solution when the space constraint is soft (see Section 4.6.1). This is an important factor in the selection of the algorithm, since it removes the requirement of a hard space constraint, and gives the user the flexibility to choose among a range of slightly different space requirements which may lead to solutions with significant variations in the benefit values.

The distinction among the greedy algorithms is quite clear between the best (*GrBenSp* and *GrSp*) and the worst (*GrBen* and *GrComp*) performers (see Figures 4.8 and 4.9). However, it is not clear whether *GrBenSp* or *GrSp* is a better choice, since they both have

the same time complexity and provide solutions of similar quality.

In order to answer the above question we have to turn our attention to the analysis presented in Section 4.6.3. This analysis shows that three of the greedy approaches we examined, including *GrSp*, may perform arbitrarily poorly under certain circumstances. The same does not seem to be true for the *GrBenSp* algorithm. We believe that *GrBenSp* is a better choice in this sense, since it avoids making poor decisions that lead to solutions very far from optimal.

Finally, we should note that *Tabu* can in many cases improve on the solution provided by the greedy algorithms. Therefore, it is beneficial to run this algorithm when the time allows it.

4.7 Related Work

In the problem of view selection for data warehouses [HRU96], we want, given a space constraint, to materialize a set of views in order to minimize the response time of queries to the data warehouse. In this case, however, by selecting one of the views required by a query we get a fraction of the associated benefit. A subsequent study shows that the heuristics for the view selection problem do not provide any competitiveness guarantee against the optimal solution [KM99], and proposes the experimental comparison of the available algorithms. Many other studies [Gup97, TS97, YKL97, BPT97] deal with the view selection problem as well.

A similar optimization problem appears in the context of database design under the name of vertical partitioning [NCWD84, NR89]. In this domain the problem can be stated as follows. We have a set of applications accessing relations in a database. Each application accesses a particular set of attributes in the relations. Each attribute requires a certain amount of space (to store all the values of the attribute in the relation), and may be used by more than one application. There is a memory hierarchy, where each level in the hierarchy has a storage capacity and different access times (e.g., main memory, local disk, network disk). We want to find an allocation of the attributes to the memory hierarchy so as to minimize the execution time of the applications. Similar to the view selection problem, partial benefits are credited, which is not appropriate for our case.

The *COSS* problem is part of a family of optimization problems known as the Weighted

Constrained Maximum Value Sub-Hypergraph problem, and the studies show that even simple instances of the problem do not yield to polynomial time solutions [NY97]. We are not aware of any algorithms proposed for the *COSS* problem in that area.

The *COSS* problem in the form that is presented here has also appeared in the domains of view selection for approximate querying in datacubes [PK01], and profile-driven data management [CFZ01b]. However, the solution of the optimization problem was not studied. All the algorithms described in this paper can be applied for the solution of the problem in the above domains.

4.8 Conclusions

Space constrained optimization problems are still significant despite the advances in storage technologies. The tremendous amount of information produced every day, as well as the limited capacities of certain mobile devices, confirm the need for efficient solutions to such optimization problems.

In this chapter we focus on the specific problem of *COSS*, where benefit values are associated with *sets* of items, instead of individual items. This optimization problem appears in various domains, such as data warehouses and pervasive computing. We derive the complexity of this problem, and propose several algorithms for its solution.

We present an experimental evaluation of the algorithms that illustrates the relative performance of the different approaches, and demonstrates the scalability of the greedy solutions. We also identify cases where some of the greedy algorithms perform poorly, and experimentally demonstrate that *GrBenSp* is a practical solution for the *COSS* problem.

Chapter 5

Case Study

In this chapter, we demonstrate how the techniques presented in Chapters 3 and 4 can be combined in a single case study over a real dataset. Namely, we start by considering a workload of queries, which are the input to the *COSS* problem, as described in Chapter 4. The output of the *COSS* problem is the set of selected queries. These queries specify the regions of the entire dataset that will be estimated with the maximum accuracy. To this end, we materialize the highest order marginals that correspond to each one of these regions, according to the discussion in Chapter 3. Then, in order to be able to answer any other query that asks for values in the dataset that fall outside the selected regions, we will also materialize the lowest order marginals (i.e., marginals of order one) for the entire dataset.

Evidently, since we are targeting the set of selected regions in the dataset, we expect to achieve better accuracy in estimating any query that asks for values inside those regions. Nevertheless, by storing the marginals of order one for the entire dataset as well, we are able to answer approximately any query, regardless of which values in the dataset it involves.

5.1 Description of Experiments

5.1.1 Dataset

For all the experiments in the case study described in this chapter we used the *census_50K* dataset, which is a multidimensional dataset from the U.S. Census Bureau. It contains four dimension attributes, namely, information about age, education, command of English, and number of children of individuals. The measure attribute is the income of the individuals.

The size of the dataset is 50,000 tuples. Table 5.1 summarizes the statistical properties of this dataset.

dataset	*min*	*max*	*mean*	*std.dev.*	*skew*
census_50K	1000.00	196623.00	24879.75	23316.54	3.62

Table 5.1: The statistical properties (min, max, mean, standard deviation, and skew) for the real dataset *census_50K*.

5.1.2 Query Workload

In this case study, we use synthetically generated query workloads in order to specify selected regions in the dataset[1]. All the regions defined by the queries in the workload are 4-dimensional hyper-rectangles. The query workloads are generated according to the following steps. These steps determine $1-$dimensional intervals that each correspond to a single side of the hyper-rectangle.

First, we set the left point of the interval by picking a number uniformly at random in the domain of the dimension attribute. Then, we draw a number from a Zipfian distribution, that, when added to the left point of the interval, determines the right end of the interval. Finally, we make sure that the interval is no longer than 1/3 the cardinality of the dimension attribute, and that it lies within the domain of the dimension attribute.

5.1.3 Error Metric

The error metric that we report in the experiments is the *Root Mean Square Error (RMSE)*, which is defined as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Y_i - \hat{Y}_i)^2}{N}},$$

where Y_i represents the original values in the dataset, $\hat{Y}_i$ the corresponding estimated values, and N is the total number of values in the dataset.

The error is computed as follows. We wish to estimate all the values in the dataset. For each value, we check whether it is contained in any of the selected regions, for which

[1] Previous work [HHK99] has considered more elaborate schemes for generating query workloads. However, these schemes were specifically developed for testing 2-dimensional indices, and are not expected to affect the outcome of our experiments.

we have stored the marginals of the highest order. If it is contained then we estimate the value based on these marginals. Otherwise, we estimate the value based on the marginals of the lowest order, which we have stored for the entire dataset. It may be the case that some value is contained in more than one selected region, and therefore, can be estimated based on different sets of marginals. We resolve these situations by choosing to estimate the value from the selected region containing this value that we encounter first. In Section 5.2 we show that this choice does not play a significant role in our experiments. Note that we estimate the values, and measure the error of the estimation for all N values of the dataset.

5.1.4 Confidence Intervals

Each of the experiments we report was run 30 times, and each time we used a different seed for the random number generator. This resulted in different sets of workloads, which in turn defined different selected regions in the dataset. All the results that we report in the following section are averages of those 30 runs. For the error metric, we also show in the graphs the corresponding confidence intervals.

The level of confidence for the values we report in the experiments is $\alpha = 0.95$, or 95%. The confidence intervals were calculated as follows. Let μ be the mean value of the series of numbers for which we wish to calculate the confidence interval. Let σ be its standard deviation, and let K be the cardinality of the series of numbers, which in our case is 30. Then, the confidence interval is defined as

$$\mu \pm Z\left(\frac{\sigma}{\sqrt{K}}\right),$$

where Z is the value corresponding to an area of $(1-\alpha)/2$ from the center of a standardized normal distribution. For $\alpha = 0.95$, the value we have picked, Z becomes $Z = 1.96$ [Kan93].

5.2 Results

5.2.1 Properties of Selected Regions

In the first set of experiments, depicted in Figure 5.1, we measure the number of tuples contained in the selected regions (depicted in light color) as a function of the space required for storing the marginals of the selected regions. The number of tuples in the graph are

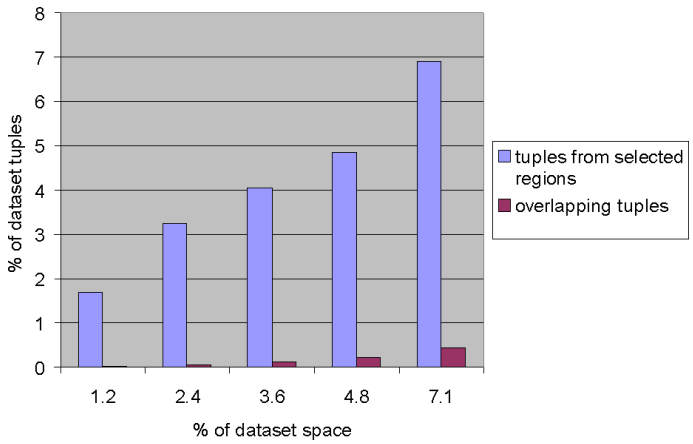


Figure 5.1: The percentage of the number of tuples in the dataset that are contained in the selected regions, and the percentage of the number of tuples in the dataset that overlap. These measures are reported as a function of the percentage of the dataset space that is occupied by the materialized marginals.

reported as a percentage of the total number of tuples in the dataset (which is 50,000), and the space occupied by the marginals as a percentage of the space required by the dataset. Each set of bars actually represents an experiment with a different set of selected regions. Note that for the five cases we report in the experiments, the selected regions in each case are a subset of those in the next larger case.

Remember that the query workload which defines the selected regions of the dataset is produced randomly, and we do not force the selected regions to be disjoint. When two, or more, selected regions overlap, this means that we have the choice to estimate the value of the measure attribute for the overlapping tuples from the information we have stored for any of these selected regions. The graph of Figure 5.1 shows that the number of tuples of the dataset that belong to more than one selected region (shown in dark color) is negligible compared to the total number of tuples in the selected regions. Therefore, we can safely assume that this parameter of the problem does not affect the outcome of our experiments significantly.

5.2.2 Queries on Detailed Values

In Figure 5.2 we depict the reduction in the error of the estimation for different sets of selected regions of increasing cardinality. The basis for computing the reduction of the

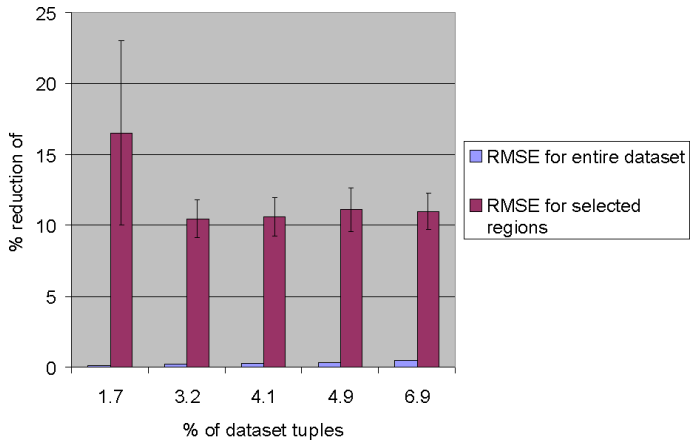


Figure 5.2: The percentage of the error reduction when estimating all the values in the dataset, and when estimating only the values contained in the selected regions. These measures are reported as a function of the percentage of the dataset tuples that is covered by the selected regions. We also depict the corresponding confidence intervals. The absolute RMSE values for the experiments shown in this graph are reported in Table 5.2.

selected regions	*queries on entire dataset*		*queries on selected regions*	
	from overall lowest	with selected highest	from overall lowest	with selected highest
1.7	19577	19557	16662	14410
3.2	19577	19536	17515	15787
4.1	19577	19522	17727	15967
4.9	19577	19509	17644	15802
6.9	19577	19482	17106	15330

Table 5.2: The absolute RMSE values for the experiments shown in the graph of Figure 5.2. These values are the means over the 30 runs of each experiment.

error are the estimates provided by the marginals of order 1, which correspond to the independence assumption. The graph shows the observed error reduction as a percentage, for both the error over the entire dataset, and the error over the values contained in the selected regions.

Evidently, the benefit of storing the marginals for the selected regions is entirely absorbed by the values contained in these selected regions. The rest of the values in the dataset do not benefit. Thus, when the reduction in the estimation error of the values in the selected regions is spread over the entire dataset, as depicted by the light colored bars in the graph of Figure 5.2, it becomes negligible.

We should note here, that the significantly higher error reduction for the first set of selected regions (leftmost dark bar in the graph) is caused by a small number of queries, whose values are approximated exceptionally well. The reduction of the error for the rest of the cases is slightly more than 10%. The fact that this error reduction remains steady across the different sets of selected regions, even though the space dedicated to the marginals is increasing, is not surprising. The additional marginals are related to a new, different, set of values in the dataset. Therefore, we should not expect them to reduce any further the estimation errors of the old set of values.

Nevertheless, as we show in Figure 5.3, the overall error is steadily diminishing when we materialize more marginals. This graph illustrates the reduction of the error for the

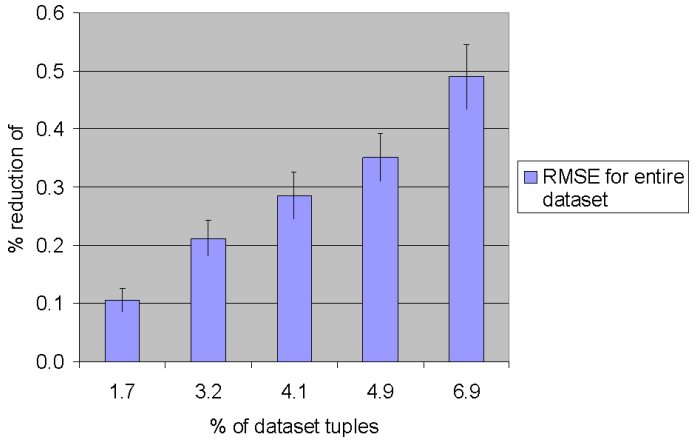


Figure 5.3: The percentage of the error reduction when estimating all the values in the dataset, as a function of the percentage of the dataset tuples that is covered by the selected regions. We also depict the corresponding confidence intervals. The absolute RMSE values for the experiments shown in this graph are reported in Table 5.3.

selected regions	*queries on entire dataset*	
	from overall lowest	with selected highest
1.7	19577	19557
3.2	19577	19536
4.1	19577	19522
4.9	19577	19509
6.9	19577	19482

Table 5.3: The absolute RMSE values for the experiments shown in the graph of Figure 5.3. These values are the means over the 30 runs of each experiment.

entire dataset as a function of the tuples covered by the selected regions. This confirms the intuition that when we use more space to store marginals in order for the selected regions to cover more tuples of the dataset, we should expect a reduction of the overall estimation error.

In the following experiments we compare the error reduction in estimating the values contained in the selected regions (Figure 5.4), and all the values in the dataset (Figure 5.5), when we employ the marginals of order 1 and order 3 of the selected regions. The marginals of order 1 are the lowest order marginals, and in the graphs are depicted with the light colored bars, while the marginals of order 3 are the highest order, and are depicted with the dark colored bars. Figure 5.4 shows that the reduction in the estimation error when

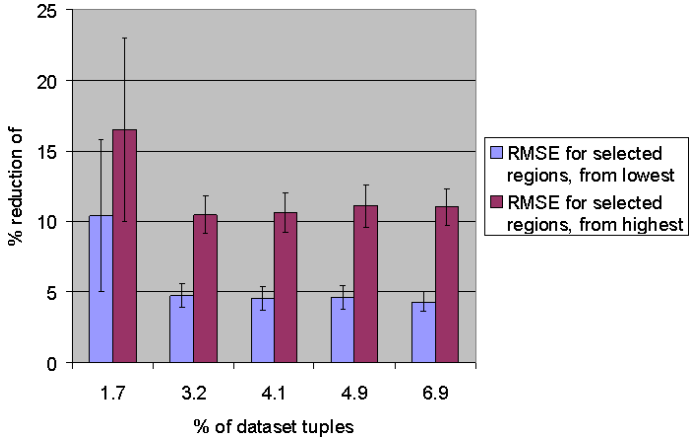


Figure 5.4: The percentage of the error reduction when estimating only the values contained in the selected regions, by using the marginals of order 1 and order 3. These measures are reported as a function of the percentage of the dataset tuples that is covered by the selected regions. We also depict the corresponding confidence intervals. The absolute RMSE values for the experiments shown in this graph are reported in Table 5.4.

we use the marginals of order 3 is up to three times larger than when using the marginals of order 1, for values in the selected regions only. When we consider all the values in the dataset, then the benefit is not as pronounced (see Figure 5.5). In this case, the error reduction is approximately twice as large when we use the marginals of higher order. This improvement in the estimation accuracy comes at the expense of space, since the marginals of order 3 require about 50 times more space than those of order 1. Clearly, we have to take into account this tradeoff when deciding whether the added accuracy, that the marginals of higher order offer, is needed.

selected regions	*queries on selected regions*		*queries on selected regions*	
	from overall lowest	with selected lowest	from overall lowest	with selected highest
1.7	16662	15311	16662	14410
3.2	17515	16739	17515	15787
4.1	17727	16982	17727	15967
4.9	17644	16885	17644	15802
6.9	17106	16416	17106	15330

Table 5.4: The absolute RMSE values for the experiments shown in the graph of Figure 5.4. These values are the means over the 30 runs of each experiment.

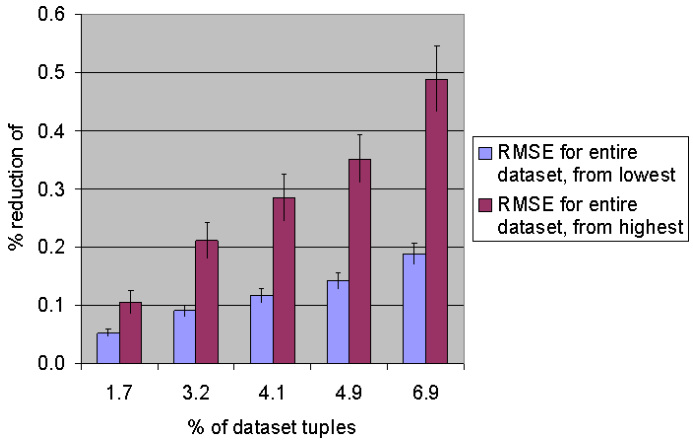


Figure 5.5: The percentage of the error reduction when estimating all the values in the dataset, by using the marginals of order 1 and order 3. These measures are reported as a function of the percentage of the dataset tuples that is covered by the selected regions. We also depict the corresponding confidence intervals. The absolute RMSE values for the experiments shown in this graph are reported in Table 5.5.

5.2.3 Aggregate Queries

In our discussion so far, we have only considered the case where we are interested in estimating all the detailed values specified by a query. The error metric has been calculated accordingly, by measuring the difference between the estimate and the real value for each one of the detailed values individually.

In the next set of experiments, we consider another important class of queries, namely, the aggregate queries. For this class of queries, we are interested in a *single* value only: the *aggregate* of all the values specified by the query. Then, the error is calculated on the estimate of the aggregate and the real aggregate value.

selected regions	*queries on entire dataset*		*queries on entire dataset*	
	from overall lowest	with selected lowest	from overall lowest	with selected highest
1.7	19577	19568	19577	19557
3.2	19577	19560	19577	19536
4.1	19577	19555	19577	19522
4.9	19577	19550	19577	19509
6.9	19577	19541	19577	19482

Table 5.5: The absolute RMSE values for the experiments shown in the graph of Figure 5.5. These values are the means over the 30 runs of each experiment.

For the following experiments, we produced a query workload of 5,000 queries, by employing the same method we used to produce the selected regions, as described earlier. We used the same sets of selected regions as before, and repeated each experiment 30 times. The results on the errors we report are over the 5,000 aggregate queries, and then averaged over the 30 repetitions of each experiment. The error metric we use is once again the RMSE.

Figure 5.6 depicts the reduction in the error when estimating aggregate queries, and when we use the marginals of order 3 for the selected regions. We measure the error

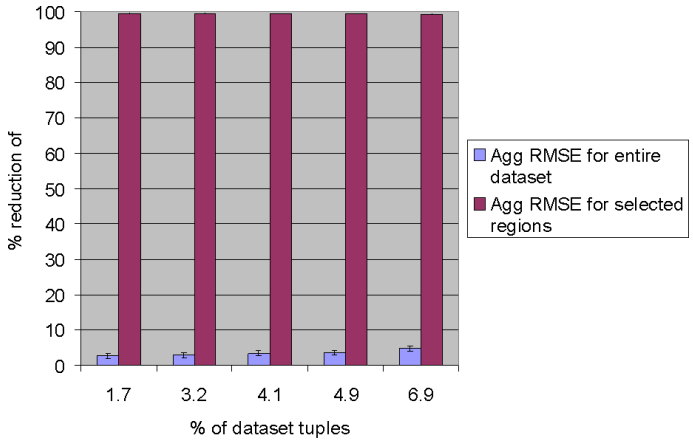


Figure 5.6: The percentage of the error reduction when estimating the aggregate value of subsets of the entire dataset, and subsets of the selected regions only. These measures are reported as a function of the percentage of the dataset tuples that is covered by the selected regions. We also depict the corresponding confidence intervals. The absolute RMSE values for the experiments shown in this graph are reported in Table 5.6.

reduction for aggregate queries over the entire dataset (light colored bars), and over the selected regions only (dark colored bars). In the latter case, we simply consider the portion

selected regions	*aggregate queries on entire dataset*		*aggregate queries on selected regions*	
	from overall lowest	with selected highest	from overall lowest	with selected highest
1.7	761268	741431	80178	327
3.2	733327	712434	97210	416
4.1	694062	670774	102492	487
4.9	670111	646311	104874	495
6.9	667404	635354	120110	755

Table 5.6: The absolute RMSE values for the experiments shown in the graph of Figure 5.6. These values are the means over the 30 runs of each experiment.

of each query that overlaps with some selected region, and we compute the aggregate value of this portion. (In each of the experiments we ran, there were approximately 1,000 queries overlapping with the selected regions.) The graph shows that the estimates produced for the aggregate queries over the selected regions is almost perfect (the error reduction is slightly less than 100%). The reason why the estimates are so accurate, is because in the general case, when estimating the values of a dataset, we expect to have an almost equal number of under-estimates and over-estimates, which will cancel each other out when we compute the aggregate value.

Even when we consider aggregate queries over the entire dataset, we still get significantly more accurate estimates. As Figure 5.7 shows, the error reduction for aggregate queries over the entire dataset is approximately 2.5%-5% for the selected regions we considered. This

selected regions	*aggregate queries on entire dataset*	
	from overall lowest	with selected highest
1.7	761268	741431
3.2	733327	712434
4.1	694062	670774
4.9	670111	646311
6.9	667404	635354

Table 5.7: The absolute RMSE values for the experiments shown in the graph of Figure 5.7. These values are the means over the 30 runs of each experiment.

reduction is 10-25 times larger than for queries on the detailed values (see Figure 5.3).

In the last experiments, we evaluate what the added benefit of using marginals of order 3 for the selected regions is, as compared to marginals of order 1. Figure 5.8 compares the

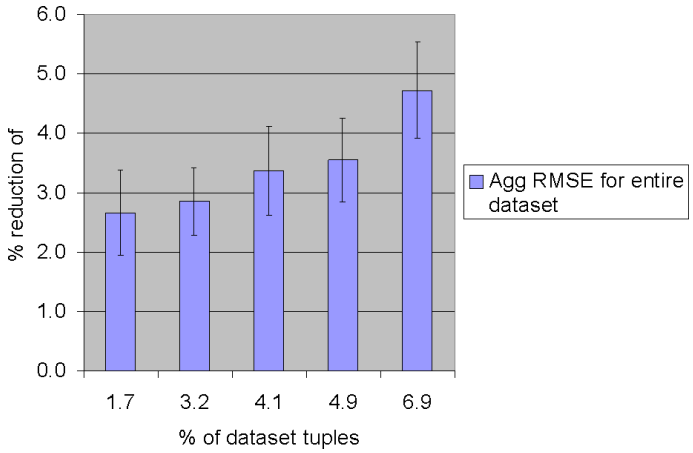


Figure 5.7: The percentage of the error reduction when estimating the aggregate value of subsets of the entire dataset. These measures are reported as a function of the percentage of the dataset tuples that is covered by the selected regions. We also depict the corresponding confidence intervals. The absolute RMSE values for the experiments shown in this graph are reported in Table 5.7.

reduction in the error of aggregate queries over the selected regions, when using marginals of order 1 (light colored bars) and order 3 (dark colored bars). The marginals of order

selected regions	*aggregate queries on selected regions*		*aggregate queries on selected regions*	
	from overall lowest	with selected lowest	from overall lowest	with selected highest
1.7	80178	18530	80178	327
3.2	97210	23093	97210	416
4.1	102492	25493	102492	487
4.9	104874	26286	104874	495
6.9	120110	32105	120110	755

Table 5.8: The absolute RMSE values for the experiments shown in the graph of Figure 5.8. These values are the means over the 30 runs of each experiment.

1 are able to reduce the estimation error by 75%, while the use of the marginals of order 3 contribute to the error reduction with an extra 25%, virtually eliminating the error in the estimation. Nevertheless, this added accuracy comes at the expense of space, since the marginals of order 3 require between one and two orders of magnitude more space than the marginals of order 1. The same error reduction is not observed when we consider aggregate queries over the entire dataset. As Figure 5.9 shows, the benefit of using marginals of higher order is in this case negligible.

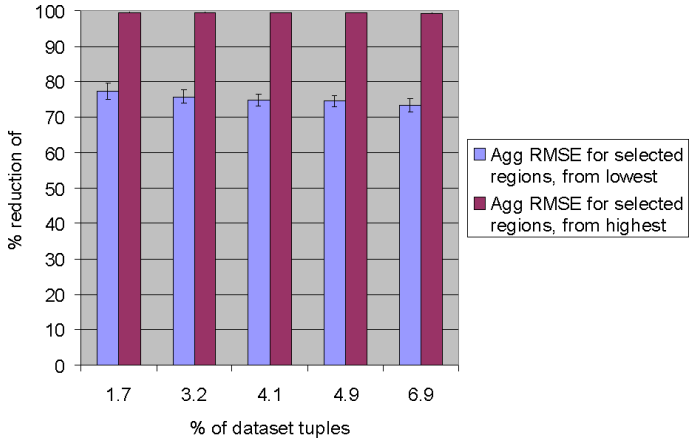


Figure 5.8: The percentage of the error reduction when estimating the aggregate value of subsets of the selected regions only, by using the marginals of order 1 and order 3. These measures are reported as a function of the percentage of the dataset tuples that is covered by the selected regions. We also depict the corresponding confidence intervals. The absolute RMSE values for the experiments shown in this graph are reported in Table 5.8.

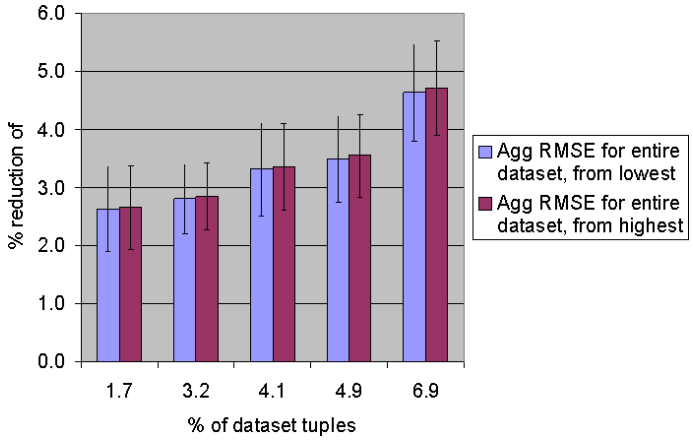


Figure 5.9: The percentage of the error reduction when estimating the aggregate value of queries over the entire dataset, by using the marginals of order 1 and order 3. These measures are reported as a function of the percentage of the dataset tuples that is covered by the selected regions. We also depict the corresponding confidence intervals. The absolute RMSE values for the experiments shown in this graph are reported in Table 5.9.

selected regions	*aggregate queries on entire dataset*		*aggregate queries on entire dataset*	
	from overall lowest	with selected lowest	from overall lowest	with selected highest
1.7	761268	741762	761268	741431
3.2	733327	712578	733327	712434
4.1	694062	670930	694062	670774
4.9	670111	646618	670111	646311
6.9	667404	635777	667404	635354

Table 5.9: The absolute RMSE values for the experiments shown in the graph of Figure 5.9. These values are the means over the 30 runs of each experiment.

5.3 Conclusions

In this chapter, we presented a case study, where we applied the techniques described in this thesis on a real dataset. In the absence of a real workload, we generated synthetic workloads, and measured the benefit of applying our techniques.

The most interesting results are illustrated by the graph in Figure 5.2. This graph indicates the extent to which a number of marginals can estimate the values of a dataset, for five different sets of selected regions of the dataset. The outcome of the experiments show that the marginals can help in the estimation of the values towards which they are targeted, whereas the benefit in the overall estimation error is minimal.

The above two distinct cases correspond to the following two extreme scenarios. The first is when the future query workload will consist of exactly the same queries as the selected regions. In this case, we will observe the error reduction bars depicted in dark color in Figure 5.2. In the second scenario, the future query workload will consist of queries spread uniformly over the entire dataset. This corresponds to the outcome of the experiments represented by the light color bars.

Obviously, the performance that we should expect from such a system lies somewhere in between those two extremes. How close the future query workload will be to the selected regions is a matter of how well we can model it, and predict it. There are cases, like the example of the census dataset, where there is no reason to believe that the future queries will be localized. On the other hand, one can think of situations where the query workload may exhibit high locality of reference. Consider the example of a consulting company that produces reports on the marketing of specific products. Then, it may only be a restricted

set of products, time periods, and market segments, that would be relevant in the analysis process.

Finally, we should note the case of aggregate queries, for which our techniques can provide estimates extremely close to the real values. In a real life scenario, we expect that the query workload will be a mixture of queries asking for the detailed values, and aggregate queries that target subsets of the regions specified by the materialized marginals (i.e., the answer to these aggregate queries is not readily available). Then, the benefit of using our framework will be even greater than what is suggested in Section 5.2.2.

Chapter 6

Discussion and Conclusions

In this chapter we present the overall conclusions from the work described in this dissertation. We also discuss future work and possible directions for research in related areas.

6.1 Conclusions

In recent years there has been a proliferation in the amount of information that is being produced. It is often times the case that not all data are stored, because of their sheer size. In order to save space people usually aggregate data and only store the aggregates. The detailed data have to be moved to tertiary storage or permanently deleted.

In this dissertation we examine the problem of using only the information contained in the aggregates in order to extract estimates of the detailed values from which the aggregates were derived. Since this is an under-specified problem, we employ the information theoretic principle of *maximum entropy*. This principle allows us to produce estimates for the detailed values based only on the aggregated, without the need to make any additional assumptions about the detailed data distribution.

Based on this framework, we describe a technique for identifying deviations in multidimensional datasets. Deviations are those values that do not follow the general trends of the dataset. Therefore, they may be interesting to the user analyst. The advantage of this technique is that it does not require any human intervention or domain specific knowledge.

We also present a method that, based on the aggregates of a multidimensional dataset, provides approximate answers to queries asking about the individual values of the dataset. It turns out that by storing a few of the largest deviations we can increase considerably the

accuracy of the estimation. In this context, we show how we can extend the framework to offer quality guarantees for the reconstruction process.

Our study indicates that the techniques we describe can capture the general trends in the data distributions we are trying to approximate. However, they are not as powerful as other techniques that have been proposed in the literature for the problem of approximate query answering, like wavelets [MVW98] [VW99], and histograms [PIHS96b] [JKM+98]. These techniques work by constructing special data structures that help in the value estimation process. They carefully examine the distribution of the data, and have the flexibility to devote more resources to those parts of the data that are harder to estimate. On the contrary, the goal of this dissertation is to evaluate the benefit of using the aggregates of a multidimensional dataset in order to estimate its detailed values. Our method does not aim to replace the above techniques of approximate query answering, and certainly cannot compete against them in the domain of problems for which they were designed. Instead, it is complementary to them, in the sense that it provides a means to produce estimates for values when the only information available are the aggregates of those values.

Finally, in this dissertation we study the above problem in a space constrained environment, where there is only a limited amount of space for storing the aggregates. In this case we have to select a subset of the aggregates, which also defines the set of queries whose values we can estimate within our framework. This is essentially a space constrained optimization problem, which we refer to as the *Constrained Set Satisfaction (COSS)* problem. The tremendous amount of information produced every day, as well as the limited capacities of certain mobile devices (either memory or bandwidth), confirm the need for efficient solutions to such optimization problems.

Since there are no known approximation algorithms for the *COSS* problem, we explore the use of greedy and randomized techniques as well as clustering based approaches. We describe an experimental evaluation of the algorithms that illustrates the relative performance of the different approaches. We also present a theoretical analysis, and identify cases where the algorithms perform poorly. This study is the first examination of practical algorithms that solve the *COSS* problem. Our experimental evaluation can serve both as a practitioner's guide, and also provide intuition about the nature of the problem from a theoretical perspective.

6.2 Future Work

There are several research directions that remain unexplored. We believe that studying them further will not only help build on the material presented in this study, but will also affect related areas of research.

We would like to explore further the problem of marginal selection. As we discussed in Chapter 3, when we build the estimated multidimensional distribution of the data we have a range of choices as to which marginals to use during this process. We may choose to use all the marginals of the highest order, or all the marginals of any other order, or even marginals from different orders. Each specific choice results in different accuracy for the estimation of the detailed values, and is associated with different space requirements. This trade off between accuracy and space can determine which choice of marginals is optimal for each situation.

In order to be able to reason about this trade off, we should have an effective and efficient way for evaluating the relative benefit of using alternative sets of marginals in the reconstruction procedure. By analyzing the data and studying their properties, we can definitely get indications as to which marginals may prove more useful. For example, a direction that seems promising is to examine the correlations [Str80] [DS86] [CDH+02] present in the data. In this case, we can choose to store those marginals that preserve the strongest correlations.

In general, we would like to be able to give guidelines for choosing marginals based on specific characteristics of the dataset. This is an important issue, because it may lead to substantial savings in the space required by the marginals with only a minimal reduction in the estimation accuracy.

Assume we have a way of evaluating the relative benefit of using different sets of marginals to estimate the answer of a query. Then, the next natural question is how to pick among those alternatives under a constraint on the space required by the marginals. This is an extension to the *COSS* problem described in Chapter 4, where we are able to satisfy each object by multiple different sets of components. In this case, the algorithms apart from selecting which objects should be satisfied, should also determine the set of

components by which each object will be satisfied. Obviously, the solution space for the problem grows significantly, as the dependencies among the components and the objects become more complex.

We now briefly describe an extension to one of the algorithms proposed for the solution of the *COSS* problem, that can be used in this new setting. The new algorithm is based on the *GrBenSp* algorithm, described in Section 4.4.

Consider the graph depicted in Figure 6.1. In this graph we illustrate an example of

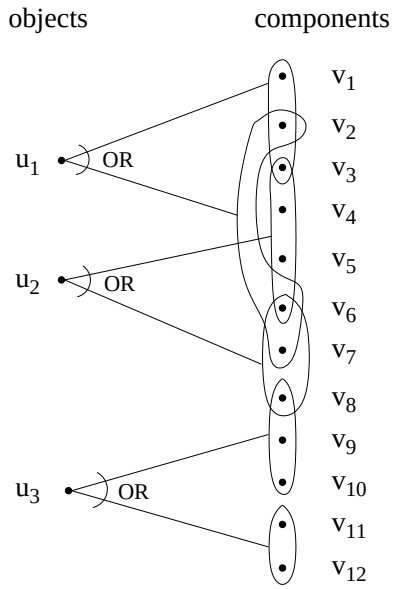


Figure 6.1: An example of the bipartite graph G that captures the dependencies among objects and components.

the dependencies that may hold among objects and their components. We denote by the closed curves the different *sets* of components. The straight lines are hyperedges between a single object and all the components in the set to which it is connected. An "OR" tag among hyperedges means that the object may be satisfied by one of the corresponding sets of components. For example, object u_1 may be satisfied either by the set of components $\{v_1, v_2, v_3\}$, or by the set $\{v_2, v_6, v_7\}$. For each object, the algorithm will select the best alternative, based on the benefit per space ratio. Then, the algorithm will select the object with the best benefit per space ratio overall.

More specifically, the outline of the algorithm is as follows. Initially, no components are selected, and therefore, no objects are satisfied. First, we compute, for a particular object,

the benefit per space ratio for each set of components that satisfies this object. During this computation we only account for the space of the components that have not been already selected. We mark the set of components that yields the largest benefit per space ratio for the particular object. We repeat the same computation for all the objects that are not currently satisfied. At the end of this process we have marked for each object a preferred set of components, and we know the corresponding benefit per space ratios. Then, among those sets of components we select the one with the highest ratio. This concludes the first iteration, and we repeat the entire process until there is no more space available.

We should note though, that in this new context the *COSS* problem presents much harder challenges, and should be studied anew.

In this dissertation we present a preliminary theoretical analysis for the greedy algorithms proposed for the solution of the *COSS* problem (see Section 4.6.3). This analysis is useful, because it indicates that the performance of some of the algorithms can become arbitrarily poor in the worst case. Nevertheless, it is far from complete. A more careful study of the theoretical properties of the algorithms is needed. This will help us comprehend the intricacies of the problem, and will lead to better understanding of the behavior of the algorithms.

An interesting application of the techniques presented in this study is in the area of selectivity estimation in the context of a *Database Management System (DBMS)*. All the state of the art DBMSs gather statistics on the data they manage, which are subsequently used by the optimizer. These statistics provide invaluable information for selectivity estimation during the process of query optimization.

In order to take correct decisions, the optimizer needs to know the joint distribution of values for a combination of attributes in the database. However, it is typical that statistical information is available only for a few, low dimensional, joint distributions. The reason for this choice is that there exist a huge number of possible joint distributions, and their space requirements are considerable.

Our techniques offer a feasible and efficient alternative. They have the ability to approximate joint distributions of high dimensionalities, based on a number of distributions of lower dimensionality. The nice property of these techniques is that they are bound to

yield more accurate estimates than the ones used by the state of the art databases, whose estimates are based on 1-dimensional histograms. Moreover, the algorithms we describe in this dissertation can effectively use any information about the distributions they are trying to approximate that is available to the system. Hints, or even exact values for parts of the unknown distributions may be available by examining the execution of the query workload [SLMK01] [BC02]. In such cases, we can use this information to improve the accuracy of the estimation of the unknown distribution.

Bibliography

[AAD+96] Sameet Agarwal, Rakesh Agrawal, Prasad Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi. On the Computation of Multidimensional Aggregates. In *VLDB International Conference*, pages 506–521, Mumbai (Bombay), India, September 1996.

[AAR96] Andreas Arning, Rakesh Agrawal, and Prabhakar Raghavan. A Linear Method for Deviation Detection in Large Databases. In *International Conference on Knowledge Discovery and Data Mining*, pages 164–169, Portland, OR, USA, August 1996.

[ACN00] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated Selection of Materialized Views and Indexes in SQL Databases. In *VLDB International Conference*, pages 496–505, Cairo, Egypt, September 2000.

[AGPR99] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. Join Synopses for Approximate Query Answering. In *ACM SIGMOD International Conference*, pages 275–286, Philadelphia, PA, USA, June 1999.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining Association Rules between Sets of Items in Large Databases. In *ACM SIGMOD International Conference*, pages 207–216, Washington, DC, USA, May 1993.

[AM92] Soraya Abad-Mota. Approximate Query Processing with Summary Tables in Statistical Databases. In *International Conference on Extending Database Technology*, pages 499–515, Vienna, Austria, March 1992.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In *VLDB International Conference*, pages 487–499, Santiago de Chile, Chile, September 1994.

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining Sequential Patterns. In *International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, March 1995.

[BC02] Nicolas Bruno and Surajit Chaudhuri. Exploiting Statistics on Query Expressions for Optimization. In *ACM SIGMOD International Conference*, pages 263–274, Madison, WI, USA, June 2002.

[BDF+97] Daniel Barbará, William DuMouchel, Christos Faloutsos, Peter J. Haas, Joseph M. Hellerstein, Yannis Ioannidis, H. V. Jagadish, Theodore Johnson, Raymond Ng, Viswanath Poosala, Kenneth A. Ross, and Kenneth C. Sevcik. The New Jersey Data Reduction Report. *Bulletin of the Technical Committee on Data Engineering*, 20(4):3–45, 1997.

[Ber82] D. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 1982.

[BFH75] Yvonne M. M. Bishop, Stephen E. Fienberg, and Paul W. Holland. *Discrete Multivariate Analysis: Theory and Practice*. The MIT Press, 1975.

[BKNS00] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: Identifying Density-Based Local Outliers. In *ACM SIGMOD International Conference*, pages 21–32, Dallas, TX, USA, May 2000.

[BL94] V. Barnet and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, Inc., 1994.

[BMUT97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. In *ACM SIGMOD International Conference*, pages 255–264, Tuscon, AZ, USA, June 1997.

[BPP96] A. Berger, S. Pietra, and V. Pietra. A Maximum Entropy Approach to Natural Language Modelling. *Computational Linguistics, 22(1)*, May 1996.

[BPT97] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized Views Selection in a Multidimensional Database. In *VLDB International Conference*, pages 156–165, Athens, Greece, September 1997.

[BS97] Daniel Barbará and Mark Sullivan. Quasi-Cubes: Exploiting Approximations in Multidimensional Databases. *ACM SIGMOD Record*, 26(3):12–17, 1997.

[BW00] Daniel Barbará and Xintao Wu. Using Loglinear Models to Compress Datacubes. In *Web-Age Information Management*, pages 311–322, Shanghai, China, June 2000.

[CCS93] E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP to User-Analysts: An IT Mandate. http://www.hyperion.com, 1993.

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An Overview of Data Warehousing and OLAP Technology. *Sigmod Record*, 26(1):65–74, 1997.

[CDH+02] Yixin Chen, Guozhu Dong, Jiawei Han, Benjamin W. Wah, and Jianyong Wang. Multi-Dimensional Regression Analysis of Time-Series Data Streams. In *VLDB International Conference*, pages 323–334, Hong Kong, China, August 2002.

[CFZ01a] Mitch Cherniack, Michael J. Franklin, and Stan Zdonik. Expressing User Profiles for Data Recharging. *IEEE Personal Communications*, pages 6–13, August 2001.

[CFZ01b] Mitch Cherniack, Michael J. Franklin, and Stanley B. Zdonik. Data Management for Pervasive Computing. In *VLDB International Conference*, page Tutorial, Rome, Italy, September 2001.

[CGN02] Surajit Chaudhuri, Ashish Gupta, and Vivek R. Narasayya. Compressing SQL Workloads. In *ACM SIGMOD International Conference*, pages 488–499, Madison, WI, USA, June 2002.

[CHY96] Ming-Syan Chen, Jiawei Han, and Philip S. Yu. Data Mining: An Overview from Database Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, 1996.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* New York: McGraw-Hill, 2001.

[CR99] S. F. Chen and R. Rosenfeld. A Gaussian Prior For Smoothing Maximum Entropy Models. *Technical Report CMU-CS-99-108, Carnegie Mellon University*, February 1999.

[CSD98] Soumen Chakrabarti, Sunita Sarawagi, and Byron Dom. Mining Surprising Patterns Using Temporal Description Length. In *VLDB International Conference*, pages 606–617, New York, NY, USA, August 1998.

[CT91] Thomas Cover and Joy Thomas. *Elements of Information Theory.* Wiley, 1991.

[DS40] W. E. Deming and F. F. Stephan. On a Least Square Adjustment of a Sampled Frequency Table When the Expected Marginal Totals Are Known. *Annals of Mathematical Statistics*, 11:427–444, 1940.

[DS86] Ralph B. D'Agostino and Michael A. Stephens. *Goodness-of-Fit Techniques.* New York: M. Dekker, 1986.

[Fay98] Usama Fayyad. Mining Databases: Towards Algorithms for Knowledge Discovery. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 21(1), 1998.

[FJS97a] C. Faloutsos, H. V. Jagadish, and N. Sidiropoulos. Recovering Information from Summary Data. *VLDB, Athens, Greece*, pages 36–45, August 1997.

[FJS97b] Christos Faloutsos, H. V. Jagadish, and N. D. Sidiropoulos. Recovering Information from Summary Data. In *VLDB International Conference*, Athens, Greece, August 1997.

[FPSM91] William J. Frawley, Gregory Piatetsky-Shapiro, and Chistopher J. Matheus. Knowledge Discovery in Databases: An Overview. *Gregory Piatetsky-Shapiro and William J. Frawley editors, Knowledge Discovery in Databases*, pages 1–27, 1991.

[GBLP96] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and

Sub-Totals. In *International Conference on Data Engineering*, pages 152–159, New Orleans, LO, USA, March 1996.

[GJ79] Michael Garey and David Johnson. *Computers and Intractability.* W. H. Freeman, 1979.

[GL97] Fred Glover and Manuel Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.

[GMW81] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* New York: Academic Press, 1981.

[GS00] Jim Gray and Prashant Shenoy. Rules of Thumb in Data Engineering. In *International Conference on Data Engineering*, San Diego, CA, USA, February 2000.

[Gup97] Himanshu Gupta. Selection of Views to Materialize in a Data Warehouse. In *ICDT International Conference*, pages 98–112, Delphi, Greece, January 1997.

[Han97] Jiawei Han. OLAP Mining: An Integration of OLAP with Data Mining. In *IFIP Conference on Data Semantics*, pages 1–11, Leysin, Switzerland, October 1997.

[Han98] Jiawei Han. Towards On-Line Analytical Mining in Large Databases. *ACM Sigmod Record*, 27(1):97–107, 1998.

[HHK99] Joseph M. Hellerstein, Lisa Hellertein, and George Kollios. On the Generation of 2-Dimensional Index Workloads. In *ICDT International Conference*, pages 113–130, Jerusalem, Israel, January 1999.

[HK00] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2000.

[HRU96] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implementing Data Cubes Efficiently. In *ACM SIGMOD International Conference*, pages 205–216, Montreal, Canada, June 1996.

[IP95] Y. Ioannidis and Viswanath Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. *ACM SIGMOD, San Jose, CA*, pages 233–244, June 1995.

[IW87] Yannis E. Ioannidis and Eugene Wong. Query Optimization by Simulated Annealing. In *ACM SIGMOD International Conference*, pages 9–22, San Francisco, CA, USA, May 1987.

[JAMS89] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by Simulated Annealing: An Experimental Evaluation. *Operations Research*, 37(6):865–892, 1989.

[JKM+98] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal Histograms with Quality Guarantees. In *VLDB International Conference*, pages 275–286, New York, NY, USA, August 1998.

[JKM99] H. V. Jagadish, Nick Koudas, and S. Muthukrishnan. Mining Deviants in a Time Series Database. In *VLDB International Conference*, pages 102–113, Edinburgh, Scotland, September 1999.

[JLS99] H. V. Jagadish, Laks V. S. Lakshmanan, and Divesh Srivastava. Snakes and Sandwiches: Optimal Clustering Strategies for a Data Warehouse. In *ACM SIGMOD International Conference*, pages 37–48, Philadephia, PA, USA, June 1999.

[JM00] Christopher Jermaine and Renée J. Miller. Approximate Query Answering in High-Dimensional Data Cubes. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Dallas, TX, USA, May 2000.

[JMS95] H. V. Jagadish, I. S. Mumick, and A. Silberschatz. View Maintenance Issues in the Chronicle Data Model. *ACM PODS*, pages 113–124, June 1995.

[Kan93] Gopal K. Kanji. *100 statistical tests*. Sage Publications, 1993.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[KK92] J. N. Kapur and H. K. Kesavan. *Entropy Optimization Principles with Applications*. Academic Press Inc, 1992.

[KM99] Howard J. Karloff and Milena Mihail. On the Complexity of the View-Selection Problem. In *ACM PODS International Conference*, pages 167–173, Philadelphia, PA, USA, May 1999.

[KMR+94] Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, Hannu Toivonen, and A. Inkeri Verkamo. Finding Interesting Rules from Large Sets of Discovered Association Rules. In *ACM International Conference on Information and Knowledge Management*, pages 401–407, Gaithersburg, MD, USA, November 1994.

[KN98] Edwin M. Knorr and Raymond T. Ng. Algorithms for Mining Distance-Based Outliers in Large Datasets. In *VLDB International Conference*, pages 392–403, New York, NY, USA, August 1998.

[KN99] Edwin M. Knorr and Raymond T. Ng. Finding Intensional Knowledge of Distance-Based Outliers. In *VLDB International Conference*, pages 211–222, Edinburgh, Scotland, September 1999.

[KR99] Yannis Kotidis and Nick Roussopoulos. DynaMat: A Dynamic View Maintenance System for Data Warehouses. In *ACM SIGMOD International Conference*, Philadelphia, PA, USA, June 1999.

[Kul68] Solomon Kullback. *Information Theory and Statistics*. John Wiley and Sons, 1968.

[LHC97] Bing Liu, Wynne Hsu, and Shu Chen. Discovering Conforming and Unexpected Classification Rules. In *Workshop on Intelligent Data Analysis in Medicine and Pharmacology*, Nagoya, Japan, August 1997.

[LL02] Guy M. Lohman and Sam S. Lightstone. SMART: Making DB2 (More) Autonomic. In *VLDB International Conference*, pages 877–879, San Diego, CA, USA, August 2002.

[LPH02] Laks V. S. Lakshmanan, Jian Pei, and Jiawei Han. Quotient Cube: How to Summarize the Semantics of a Data Cube. In *VLDB International Conference*, pages 778–789, Hong Kong, China, August 2002.

[LSPC00] Wolfgang Lehner, Richard Sidle, Hamid Pirahesh, and Roberta Cochrane. Maintenance of Cube Automatic Summary Tables. In *ACM SIGMOD International Conference*, pages 512–513, Dallas, TX, USA, May 2000.

[Mal93] F. Malvestuto. A Universal Scheme Approach to Statistical Databases Containing Homogeneous Summary Tables. *ACM TODS, 18(4)*, pages 678–708, December 1993.

[Man99] Heikki Mannila. Theoretical Frameworks for Data Mining. Invited talk in ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, May 1999.

[MQM97] Inderpal Singh Mumick, Dallan Quass, and Barinderpal Singh Mumick. Maintenance of Data Cubes and Summary Tables in a Warehouse. In *ACM SIGMOD International Conference*, pages 100–111, Tuscon, AZ, USA, June 1997.

[MRL99] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Datasets. In *ACM SIGMOD International Conference*, pages 251–262, Philadephia, PA, USA, June 1999.

[MSW72] W. T. McCormick, P. J. Schweitzer, and T. W. White. Problem Decomposition and Data Reorganization by a Clustering Technique. *Operations Research*, 20(5):993–1009, 1972.

[MVW98] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Wavelet-Based Histograms for Selectivity Estimation. In *ACM SIGMOD International Conference*, pages 448–459, Seattle, WA, USA, June 1998.

[MVW00] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Dynamic Maintenance of Wavelet-Based Histograms. In *VLDB International Conference*, pages 101–110, Cairo, Egypt, September 2000.

[NCWD84] Shamkant Navathe, Stefano Ceri, Gio Wiederhold, and Jinglie Dou. Vertical Partitioning Algorithms for Database Design. *ACM Transactions on Database Systems*, 9(4):680–710, 1984.

[NLHP98] Raymond T. Ng, Laks V. S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory Mining and Pruning Optimizations of Constrained Associations Rules. In *ACM SIGMOD International Conference*, pages 13–24, Seattle, WA, USA, June 1998.

[NR89] Shamkant B. Navathe and Minyoung Ra. Vertical Partitioning for Database Design: A Graphical Algorithm. In *ACM SIGMOD International Conference*, pages 440–450, Portland, OR, USA, May 1989.

[NY97] David Nehme and Gang Yu. The Cardinality and Precedence Constrained Maximum Value Sub-Hypergraph Problem and its Applications. *Discrete Applied Mathematics*, 74:57–68, 1997.

[Pal00] Themistoklis Palpanas. Knowledge Discovery in Data Warehouses. *ACM SIGMOD Record*, 29(3):88–100, 2000.

[PCY95] Jong-Soo Park, Ming-Syan Chen, and Philip S. Yu. An Effective Hash Based Algorithm for Mining Association Rules. In *ACM SIGMOD International Conference*, pages 175–186, San Jose, CA, USA, May 1995.

[PG99] Viswanath Poosala and Venkatesh Ganti. Fast Approximate Answers to Aggregate Queries on a Data Cube. In *International Conference on Scientific and Statistical Database Management*, pages 24–33, Cleveland, OH, USA, 1999.

[PI97] Viswanath Poosala and Yannis E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *VLDB International Conference*, pages 486–495, Athens, Greece, August 1997.

[PIHS96a] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. *ACM SIGMOD, Montreal Canada*, pages 294–305, June 1996.

[PIHS96b] Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, and Eugene J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. In *ACM SIGMOD International Conference*, pages 294–305, Montreal, Canada, June 1996.

[PK01] Themistoklis Palpanas and Nick Koudas. Entropy Based Approximate Querying and Exploration of Datacubes. In *International Conference on Scientific and Statistical Database Management*, pages 81–90, Fairfax, VA, USA, July 2001.

[PSCP02] Themistoklis Palpanas, Richard Sidle, Roberta Cochrane, and Hamid Pirahesh. Incremental Maintenance for Non-Distributive Aggregate Functions. In *VLDB International Conference*, Hong Kong, China, August 2002.

[PT98] Balaji Padmanabhan and Alexander Tuzhilin. A Belief-Driven Method for Discovering Unexpected Patterns. In *International Conference on Knowledge Discovery and Data Mining*, pages 94–100, New York, NY, USA, August 1998.

[RKR97] Nick Roussopoulos, Yannis Kotidis, and Mema Roussopoulos. Cubetree: Organization of and Bulk Incremental Updates on the Data Cube. In *ACM SIGMOD International Conference*, pages 89–99, Tuscon, AZ, USA, June 1997.

[RRS00] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient Algorithms for Mining Ouliers from Large Data Sets. In *ACM SIGMOD International Conference*, pages 427–438, Dallas, TX, USA, May 2000.

[RS97] Kenneth A. Ross and Divesh Srivastava. Fast Computation of Sparse Datacubes. In *VLDB International Conference*, pages 116–125, Athens, Greece, September 1997.

[SA95] Ramakrishnan Srikant and Rakesh Agrawal. Mining Generalized Association Rules. In *VLDB International Conference*, pages 407–419, Zurich, Switzerland, September 1995.

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *International Conference on Extending Database Technology*, pages 3–17, Avignon, France, March 1996.

[SAM98] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. Discovery-driven Exploration of OLAP Data Cubes. In *International Conference on Extending Database Technology*, pages 168–182, Valencia, Spain, March 1998.

[Sar97] Sunita Sarawagi. Indexing OLAP Data. *IEEE Data Engineering Bulletin*, 20(1):36–43, 1997.

[Sar99] Sunita Sarawagi. Explaining Differences in Multidimensional Aggregates. In *VLDB International Conference*, pages 42–53, Edinburgh, Scotland, September 1999.

[Sar00] Sunita Sarawagi. User-Adaptive Exploration of Multidimensional Data. In *VLDB International Conference*, pages 307–316, Cairo, Egypt, September 2000.

[SDNR96] Amit Shukla, Prasad Deshpande, Jeffrey F. Naughton, and Karthikeyan Ramasamy. Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies. In *VLDB International Conference*, pages 522–531, Mumbai (Bombay), India, September 1996.

[SFB99] Jayavel Shanmugasundaram, Usama M. Fayyad, and P. S. Bradley. Compressed Data Cubes for OLAP Aggregate Query Approximation on Continuous Dimensions. In *International Conference on Knowledge Discovery and Data Mining*, pages 223–232, San Diego, CA, USA, August 1999.

[SLMK01] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. LEO - DB2's LEarning Optimizer. In *VLDB International Conference*, pages 19–28, Rome, Italy, September 2001.

[SON95] Ashok Savasere, Edward Omiecinski, and Shamkant Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. In *VLDB International Conference*, pages 432–444, Zurich, Switzerland, September 1995.

[SS94] Sunita Sarawagi and Michael Stonebraker. Efficient Organization of Large Multidimensional Arrays. In *International Conference on Data Engineering*, pages 328–336, Houston, TX, USA, February 1994.

[SS01] Gayatri Sathe and Sunita Sarawagi. Intelligent Rollups in Multidimensional OLAP Data. In *VLDB International Conference*, pages 531–540, Rome, Italy, September 2001.

[ST96a] A. Silberschatz and A. Tuzhilin. What Makes Patterns Interesting in Knowledge Discovery Systems. *IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No 6.*, pages 970–974, December 1996.

[ST96b] Avi Silberschatz and Alexander Tuzhilin. User-Assisted Knowledge Discovery: How Much Should the User Be Involved. In *ACM-SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, Montreal, ON, Canada, June 1996.

[ST96c] Avi Silberschatz and Alexander Tuzhilin. What Makes Patterns Interesting in Knowledge Discovery Systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):970–974, May 1996.

[Str80] Gilbert Strang. *Linear Algebra and its Applications.* Academic Press, 1980.

[Toi96] Hannu Toivonen. Sampling Large Databases for Association Rules. In *VLDB International Conference*, pages 134–145, Mumbai (Bombay), India, September 1996.

[TS97] Dimitri Theodoratos and Timos K. Sellis. Data Warehouse Configuration. In *VLDB International Conference*, pages 126–135, Athens, Greece, September 1997.

[TUA+98] Dick Tsur, Jeffrey D. Ullman, Serge Abiteboul, Chris Clifton, Rajeev Motwani, Svetlozar Nestorov, and Arnon Rosenthal. Query Flocks: A Generalization of Association-Rule Mining. In *ACM SIGMOD International Conference*, Seattle, WA, USA, June 1998.

[VW99] Jeffrey Scott Vitter and Min Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. In *ACM SIGMOD International Conference*, pages 193–204, Philadelphia, PA, USA, June 1999.

[VWI98] Jeffrey Scott Vitter, Min Wang, and Bala Iyer. Data Cube Approximation and Histograms via Wavelets. In *ACM International Conference on Information and Knowledge Management*, pages 96–104, Washington, DC, USA, 1998.

[YKL97] Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for Materialized View Design in Data Warehousing Environment. In *VLDB International Conference*, pages 136–145, Athens, Greece, September 1997.

[ZCL+00] Markos Zaharioudakis, Roberta Cochrane, George Lapis, Hamid Pirahesh, and Monica Urata. Answering Complex SQL Queries Using Automatic Summary Tables. In *ACM SIGMOD International Conference*, pages 105–116, Dallas, TX, USA, May 2000.

[ZDN97] Yihong Zhao, Prasad Deshpande, and Jeffrey F. Naughton. An Array-Based Algorithm for Simultaneous Multidimensional Aggregates. In *ACM SIGMOD International Conference*, pages 159–170, Tuscon, AZ, USA, June 1997.

[ZXH98] Osmar R. Zaïane, Man Xin, and Jiawei Han. Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs. In *Advances in Digital Libraries*, pages 19–29, Santa Barbara, CA, USA, April 1998.