Interpretable and Informative Explanations of Outcomes

Kareem El Gebaly
University of Waterloo
kelgebal@uwaterloo.ca

Parag Agrawal [*]
Twitter, Inc.
paraga@twitter.com

Lukasz Golab
University of Waterloo
lgolab@uwaterloo.ca

Flip Korn [*]
Google Research
flip@google.com

Divesh Srivastava
AT&T Labs–Research
divesh@research.att.com

ABSTRACT

In this paper, we solve the following data summarization problem: given a multi-dimensional data set augmented with a binary attribute, how can we construct an interpretable and informative summary of the factors affecting the binary attribute in terms of the combinations of values of the dimension attributes? We refer to such summaries as explanation tables. We show the hardness of constructing optimally-informative explanation tables from data, and we propose effective and efficient heuristics. The proposed heuristics are based on sampling and include optimizations related to computing the information content of a summary from a sample of the data. Using real data sets, we demonstrate the advantages of explanation tables compared to related approaches that can be adapted to solve our problem, and we show significant performance benefits of our optimizations.

1. INTRODUCTION

Many analytics make use of data augmented with binary attributes, corresponding to, e.g., the satisfaction of a predicate or the outcome of a hypothesis being tested on individual data records. Applications include identifying target groups for STEM intervention (e.g., `age=teen AND gender=female`) and identifying risk factors in epidemiology (e.g., `weight=obese AND habit=smoking`). Before performing complex statistical analysis and decision support, users often wish to explore and summarize the data. In this paper, we introduce the concept of *Explanation Tables*, which are concise summaries of multi-dimensional relations with a binary outcome attribute. The goal of an explanation table is to provide an interpretable and informative summary of the factors affecting the outcome attribute. We propose an information-theoretic framework for computing informative explanation tables, and we present novel algorithms that significantly improve the efficiency of explanation table construction compared to straightforward approaches.

To motivate explanation tables, consider Table 1, which shows an athlete's exercise log, with each row containing an id, the time and

[*]Work done while the authors were at AT&T Labs - Research.

Table 1: An Exercise relation

id	day	time	meal	goal met?
1	Fri	Dawn	Banana	Yes
2	Fri	Night	Green salad	Yes
3	Sun	Dusk	Oatmeal	Yes
4	Sun	Morning	Banana	Yes
5	Mon	Afternoon	Oatmeal	Yes
6	Mon	Midday	Banana	Yes
7	Tue	Morning	Green salad	No
8	Wed	Night	Burgers	No
9	Thu	Dawn	Oatmeal	Yes
10	Sat	Afternoon	Nuts	No
11	Sat	Dawn	Banana	No
12	Sat	Dawn	Oatmeal	No
13	Sat	Dusk	Rice	No
14	Sat	Midday	Toast	No

Table 2: An explanation table over the Exercise relation

day	time	meal	goal met=Yes?	count
*	*	*	.5	14
Sat	*	*	0	5
*	*	Banana	.75	4
*	*	Oatmeal	.75	4

day of the week of the exercise, the meal eaten before the exercise, and a binary attribute indicating the outcome of the exercise, i.e., whether a target goal was met. Table 2 illustrates a corresponding explanation table. It contains a list of patterns that specify subsets of tuples; patterns consist of attribute values or the wildcard symbol "*" denoting all possible values. Each pattern is associated with the number of tuples it covers (count) and a fraction indicating how many of these tuples have the binary outcome attribute equal to Yes (or 1 or True). The first pattern in Table 2 matches all the tuples in Table 1 and indicates that, on average, half the workouts were successful. The second pattern reveals that all Saturday workouts were unsuccessful, while the last two patterns state that eating bananas or oatmeal led to successful workouts 75 percent of the time.

In order to be useful in practice, we argue that explanation tables must satisfy three conditions. First, they must be *interpretable*, i.e., able to reveal macro-level trends. In real data, trends often overlap, such as the Saturday and Banana patterns in Table 2. Thus, we must allow *overlapping* patterns to make it easy to piece together the salient factors affecting the outcome attribute. The second condition is that explanation tables must be *informative*, i.e., given the dimension attribute values, the explanation table should enable micro-level reconstruction of the binary attribute with measurable accuracy. The third condition is that explanation tables must be *efficient* to construct, even on large data sets.

1.1 Challenges and Contributions

It may appear that existing approaches can be applied to generate explanation tables. However, existing work addresses either interpretability or informativeness, but not both. For example, classification algorithms such as decision trees are informative, but the rules they encode do not overlap (otherwise, it would not be obvious which rule to use when making a prediction), and therefore they are not interpretable according to our criteria. On the other hand, there are various database summarization techniques that generate overlapping patterns and are interpretable [5, 6, 9]. However, these techniques optimize for different objectives than informativeness (e.g., minimum cardinality: covering all the tuples with outcome 1 using the fewest possible patterns).

In this paper, we show how to efficiently generate explanation tables that are both interpretable and informative. We employ a classical information-theoretic approach to select patterns that provide the most information gain about the distribution of the outcome attribute. A similar approach was used in prior work on data cube exploration, but overlapping patterns were not supported [11]. This restriction helped make the problem much more tractable because it allows for optimizations based on partitioning. How to lift this restriction is a key technical contribution of this paper.

In particular, the challenge lies in the size of the search space of overlapping patterns. As we will show, even greedily selecting the most informative patterns is infeasible for even moderately-sized data sets. We show that straightforward sampling is not effective and we propose two solutions for making better use of samples in explanation table construction. One of the proposed algorithms, *Flashlight*, finds an efficient way to analyze the candidate patterns pertaining to a given sample, and the other, *Laserlight*, is more efficient on a given sample but needs to use larger samples to be effective.

The specific contributions of this paper are as follows.

1. We formalize the notion of explanation tables for summarizing multi-dimensional data sets associated with a binary outcome attribute.

2. We prove that constructing optimally-informative explanation tables from data is NP-hard in the number of input data tuples.

3. We present greedy heuristics for explanation table construction based on random sampling, and we show how to implement them in a database system using a sequence of views.

4. Using real data sets, we show that explanation tables are more informative than existing approaches that can be adapted to solve our problem, and we demonstrate significant performance gains of the proposed algorithms and benefits of a DBMS-based implementation, as compared to straightforward sampling techniques.

1.2 Roadmap

The remainder of this paper is organized as follows. Section 2 formally defines explanation tables and proves that constructing them is NP-hard; Section 3 discusses related work; Section 4 presents our algorithms for constructing explanation tables; Section 5 contains an experimental evaluation of the proposed algorithms as well as a comparison of explanation tables with related summarization approaches; and Section 6 concludes the paper with directions for future work.

Table 3: Symbols used in this paper

Symbol	Explanation
$\mathcal{R}$	Relational schema
D	Relation instance
t	A tuple
v	Binary outcome attribute
d	Number of attributes not including the outcome attribute
$\mathcal{P}$	Set of all possible patterns
p	A pattern
$S_D(p)$	Support set of p over D
$f_{D,v}(p)$	Fraction of tuples in D matching p that have $v = 1$
T	An explanation table
$u^*(t)$	A maximum-entropy estimate of $v(t)$ given by T
$g_{D,u^*}(p)$	Estimated fraction of tuples in D matching p that have $v = 1$ based on u^*
τ	Desired KL-Divergence threshold
s	A uniform sample of tuples from D

2. PRELIMINARIES

In this section, we present the necessary background and we formally define the problem we want to solve. Table 3 lists the symbols used in this paper.

2.1 Definitions

Consider a relation instance D over a schema $\mathcal{R}$ with attributes $A_1, A_2, ..., A_d$ and a binary outcome attribute v. Let $v : D \to \{0, 1\}$ be a function that maps each tuple $t \in D$ to its outcome. We assume that the semantics of v have some structure to them with respect to the attribute value combinations of A_1 through A_d, and we want to summarize the outcomes using these combinations.

Let $\mathcal{P}(\mathcal{R}) = (dom(A_1) \cup \{*\}) \times \cdots \times (dom(A_d) \cup \{*\})$. A *pattern* $p \in \mathcal{P}(\mathcal{R})$ is a tuple of symbols, where each symbol is either a value in the corresponding attribute's domain or a wildcard symbol '*'.

Definition 1. A tuple t *matches* a pattern p if, for each A_j in $\mathcal{R}$, either $p[A_j] =$ '*' or $t[A_j] = p[A_j]$. We denote this match using the operator "$\asymp$" as $t \asymp p$. A tuple can match many patterns and a pattern can match many tuples. For example, the pattern (Sat,*,*) from Table 2 matches all tuples with day=Sat regardless of the values of the other attributes, i.e., tuples 10-14 from Table 1.

Definition 2. The *support set* of a pattern p over a relation instance D, $S_D(p)$, is the set of tuples in D matching p, that is, $\{t \in D : t \asymp p\}$. The *support* of p is $|S_D(p)|$.

Let $\mathcal{P}(D) = \{p \in \mathcal{P}(\mathcal{R}) : S_D(p) \neq \emptyset\}$ (i.e., the datacube of D) and let the function $f_{D,v} : \mathcal{P}(D) \to [0, 1]$ give the fraction of matching tuples with outcome 1 for each pattern; that is, $f_{D,v}(p) = \frac{1}{|S_D(p)|} \sum_{t \asymp p} v(t)$.

Definition 3. An *explanation table (ET)* is a summary of outcomes given as a collection of patterns (denoted T), each pattern p in T having the associated fraction, $f_{D,v}(p)$, denoting the fraction of matching tuples where the outcome v evaluates to 1, along with its support $|S_D(p)|$. We have seen an example of an explanation table in Table 2.

We now discuss how to quantify the information content of an explanation table. The key observation is that the fractions $f_{D,v}(p)$ reported for each pattern can be used to estimate the distribution of the outcome attribute v with respect to the value combinations of the A_j's. To compute this estimate, we apply the maximum entropy principle, stating that the probability distribution which best represents the current state of knowledge, subject to known constraints, is the one with highest entropy: a distribution with lower entropy would assume information we do not possess and one with higher entropy would violate the constraints we do possess. Given an explanation table T, the maximum-entropy approximation u^* of v, over all $t \in D$, is selected from the u which maximizes

$$\sum_{t \in D} -u(t) \cdot \log(u(t)) \tag{1}$$

subject to

$$\begin{array}{ll} \forall_{t \in D} & 0 \leq u(t) \leq 1 \\ \forall_{p \in T} & \sum_{t \asymp p} u(t) = |S_D(p)| \cdot f_{D,v}(p). \end{array}$$

That is, for each tuple t in D, the estimated value of its outcome attribute must be between zero and one, and the estimates must be consistent with the fractions $f_{D,v}$ (of tuples having outcome 1) associated with the patterns in T.

For example, recall the Exercise relation from Table 1 and suppose we are given an explanation table T with only the first two patterns from Table 2, (*,*,*) and (Sat,*,*), call them p_1 and p_2, respectively. Using T to estimate v, we require that $\sum_{t \asymp p_1} u(t) = 14 \cdot 0.5 = 7$ and $\sum_{t \asymp p_2} u(t) = 5 \cdot 0 = 0$. The latter implies that all the Saturday tuples must have $u^*(t) = 0$, which means that the sum of the $u(t)$ values of all non-Saturday tuples must be seven. The maximum-entropy solution is to assign the same $u(t)$ to all nine non-Saturday tuples, therefore for all $t \in S_D(p_1) \setminus S_D(p_2)$ we set $u^*(t) = \frac{7}{9}$.

Solving Equation 1 was straightforward in the above example, but it may require complex optimization for large explanation tables. Fortunately, we can apply the classical *iterative scaling* technique. Extended to our problem, a key result from [1] states that the distribution of u^* has the following form:

$$u^*(t) = \frac{e^{\sum_{p \in T: t \asymp p} \lambda(p)}}{1 + e^{\sum_{p \in T: t \asymp p} \lambda(p)}} \tag{2}$$

where the $\lambda(p)$'s are multipliers associated with each pattern. The idea behind iterative scaling is to refine the λ values until Equation 2 gives $u^*(t)$'s that match (or converge to within a small fraction, say 0.01, of) the constraints implied by the explanation table, namely $\sum_{t \asymp p} u(t) = |S_D(p)| f_{D,v}(p)$ (see [1] for technical details). Returning to the above example, for $p_1 =$ (*,*,*) iterative scaling gives $\lambda(p_1) = 1.122$ and for $p_2 =$ (Sat,*,*) we get $\lambda(p_2) = -9$. Thus, for non-Saturday tuples, which only match p_1, we get $u^*(t) = \frac{e^{1.1822}}{1+e^{1.1822}} = 0.77$ and for Saturday tuples, which match both patterns, we get $u^*(t) = \frac{e^{1.1822-9}}{1+e^{1.1822-9}} = 0.00004$.

To quantify the goodness of an explanation table, we compute the Kullback-Leibler (KL) divergence, abbreviated $D_{KL}(v \| u^*)$, between the true distribution of v over the different value combinations of the A_j's and the maximum-entropy estimate u^* implied by the explanation table. For a given tuple t, this KL-divergence is defined as

$$D_{KL}(v(t) \| u^*(t)) = v(t) \log\left(\frac{v(t)}{u^*(t)}\right) + (1 - v(t)) \log\left(\frac{1 - v(t)}{1 - u^*(t)}\right).$$

Since the outcome attribute v is binary, $D_{KL}(v \| u^*)$, which is $\sum_{t \in D} D_{KL}(v(t) \| u^*(t))$, works out to

$$\sum_{t \in D | v(t)=1} \log\left(\frac{1}{u^*(t)}\right) + \sum_{t \in D | v(t)=0} \log\left(\frac{1}{1 - u^*(t)}\right).$$

2.2 Problem Statement and Complexity

Given a relation instance D with a binary outcome attribute v and a maximum KL-divergence threshold τ, the *Explanation Table Discovery Problem* is to find an explanation table $T \subseteq \mathcal{P}(D)$ of smallest size, with $u^*(t)$'s determined by maximum entropy as per Equation 1, such that $D_{KL}(v \| u^*) \leq \tau$. Note that there always exists a feasible solution with $D_{KL}(v \| u^*) = 0$ if we include all the possible patterns in T. However, since we want T of small size to help ensure interpretability, we need to carefully choose patterns whose $f_{D,v}$ fractions can be used to construct a good estimate of v.

We also consider a dual version of this problem, which is to find T minimizing $D_{KL}(v \| u^*)$ subject to a constraint on the number of patterns in T.

CLAIM 1. *Explanation Table Discovery is NP-hard.*

PROOF. We give a polynomial-time reduction from VERTEX COVER IN TRIPARTITE GRAPHS. Let G be a tripartite graph with a vertex partition (A, B, C) having m edges. Denote the vertices of A, B and C as a_i, b_j and c_k, respectively. Given edges $e \in G$, we populate a relation instance as follows:

- if $e = \{a_i, b_j\}$ then add a tuple (a_i, b_j, z_{ij});
- if $e = \{a_i, c_k\}$ then add a tuple (a_i, y_{ik}, c_k); and
- if $e = \{b_j, c_k\}$ then add a tuple (x_{jk}, b_j, c_k);

where each of the z_{ij}'s, y_{ik}'s and x_{jk}'s are unique. As for the outcome attribute v, assign it to be one for all these tuples. In addition, for each tuple (a_i, b_j, z_{ij}), add a tuple $(\alpha_i, \beta_j, z_{ij})$ where $\alpha_i \neq a_{i'}$ for all i and i' and $\beta_j \neq b_{j'}$ for all j and j'. Also $\alpha_i \neq \alpha_{i'}$ for all $i \neq i'$ and $\beta_j \neq \beta_{j'}$ for all $j \neq j'$. Assign $v = 0$ to all of these additional tuples. Furthermore, by symmetry, a tuple $(\alpha_i, y_{ik}, \gamma_k)$ is added for each tuple (a_i, y_{ik}, c_k) and a tuple $(x_{jk}, \beta_j, \gamma_k)$ is added for each tuple (x_{jk}, b_j, c_k), all with $v = 0$. The relation instance now contains $2m$ tuples, m of which have $v = 1$ and m of which have $v = 0$. Clearly, this reduction can be done in polynomial time.

Now consider the different possibilities for the smallest explanation table T having $D_{KL}(v \| u^*) = 0$. Let us start with the all-wildcards pattern (*,*,*), whose $f_{D,v}$ value is 0.5 since exactly half the tuples have $v = 0$ and half have $v = 1$. To get $D_{KL}(v \| u^*)$ down to zero, we need to find patterns that cover all and only those tuples with $v = 0$ or $v = 1$; these patterns will be associated with $f_{D,v}$ fractions that will allow us to infer u^*'s which correspond exactly to $v(t)$. If we want to add patterns for tuples with $v = 0$, we need m of them because of the way we constructed our relation instance: each tuple with $v = 0$ requires a separate pattern. Clearly, this cannot be any more efficient than adding patterns that cover tuples with $v = 1$, so we choose to find patterns that identify subsets of tuples with $v = 1$.

Consider a pattern p and assume by symmetry that its first component is a_i. Now replace p by $p' = (a_i, *, *)$. The new pattern covers at least as many tuples as p, all of which have $v = 1$. Furthermore, each such pattern corresponds to a vertex in a natural way. By symmetry, similar statements apply for edges $\{a_i, c_k\}$ and $\{b_j, c_k\}$. Thus, the size of the smallest vertex cover is at most the number of patterns needed to cover all the tuples with $v = 1$. As a result, an optimal answer can be obtained by choosing the all-wildcards pattern $(*, *, *)$ with $f_{D,v} = 0.5$, plus the patterns corresponding to the vertex cover (all of which will have $f_{D,v} = 1$). It follows that the minimum size of an explanation table equals the minimum size of a vertex cover plus 1. $\square$

Table 4: The first four patterns output by the SURPRISE operator over the Exercise relation

day	time	meal	goal met=Yes?	count
*	*	*	.5	14
*	*	Oatmeal	.75	4
*	*	Banana	.75	4
*	*	Nuts	1	1

3. RELATED WORK

Existing approaches that could potentially be used to construct explanation tables are either informative or interpretable, but not both. For example, rule-based classifiers and decision trees are informative since the rules are meant to be used for prediction, but they are not interpretable according to our criteria because they generally do not allow overlapping patterns (so it is obvious which rule to use for making predictions). It is well-known that expressing overlapping rules using non-overlapping techniques such as decision trees can lead to a blow-up in the number of required rules that is exponential in the number of attributes [12]. In Section 5, we will experimentally show that explanation tables are more informative than various types of decision trees of comparable size.

Similarly, the SURPRISE operator proposed in [11] for entropy-driven data exploration does not allow overlapping patterns unless one is fully contained in another. This technique allows a user to explore informative regions of a datacube compared to what the user has already seen. Patterns are chosen based on the reduction in the KL-divergence between the actual aggregate values and the maximum-entropy-estimates based on the already-seen subsets of the cube. Table 4 shows the first four patterns output by the SURPRISE operator over the Exercise relation from Table 1, formatted as an explanation table. These four patterns are less informative than those shown in Table 2 and they do not reveal the interesting information about Saturday workouts being ineffective. We will experimentally show the advantages of explanation tables over the SURPRISE operator in Section 5.

In addition to [11], there has been other work on pattern mining that uses the maximum-entropy principle. For example, [10] solves the problem of finding informative item sets that summarize a set of transactions and focuses on optimizing the iterative scaling step for their specific problem. We are not aware of any previous work on optimizing this process in the context of overlapping patterns over multi-dimensional data that we consider in this paper.

There are numerous database summarization techniques that generate overlapping patterns and are interpretable; see, e.g., [3, 4, 5, 6, 8, 9, 14]. However, these techniques optimize for different objectives such as minimum cardinality, e.g., attempting to cover most or all tuples with outcome 1 using the fewest possible patterns. In Section 5, we will experimentally demonstrate the advantages of explanation tables versus a representative algorithm from this class of approaches in terms of informativeness.

Explanation tables are also related to outlier detection. For example, the pattern (Sat,*,*) in Table 2 represents a subset of tuples with an unusually low fraction of tuples having outcome 1. There has been recent work on explaining outliers using predicates that are conceptually similar to our patterns [13]. However, it addressed a different problem using different techniques: given a subset of the results of an aggregation query that were marked by a user as outliers, identify subsets of tuples in the underlying table that most influenced these query results.

Algorithm 1 Greedy Explanation Table Construction (Baseline)

```
Input: $D, \tau$
1: $T \leftarrow \{(*, \ldots, *)\}$
2: $u^* \leftarrow f_{D,v}(*, \ldots, *)$
3: while $D_{KL}(v \parallel u^*) > \tau$ do
4:     $p_{max} \leftarrow \arg\max_{p \in \mathcal{P}(D)} gain(p)$
5:     $T \leftarrow T \cup \{p_{max}\}$
6:     update $u^*$ based on $T$ via iterative scaling
7: end while
```

4. ALGORITHMS

Since computing optimal explanation tables is NP-hard, we now propose heuristic solutions. We present the overall greedy framework in Section 4.1. We describe a straightforward sampling approach in Section 4.2 and propose two improvements, *Flashlight* and *Laserlight*, in Sections 4.3 and 4.4 respectively. We summarize the differences among the proposed algorithms in Section 4.5.

4.1 Baseline: A Basic Greedy Framework

Algorithm 1 shows the greedy approach to constructing explanation tables, which will serve as a template for all the algorithms presented in this section. Line 1 selects the all-wildcards pattern that matches the entire relation instance D and line 2 assigns a u^* estimate to every tuple equal to the fraction of 1-outcomes in all of D. The loop in lines 3-7 adds one pattern at a time into T until $D_{KL}(v \| u^*)$ drops below the given threshold. Line 4 examines the space of all possible patterns occurring in D (i.e., the datacube $\mathcal{P}(D)$) and selects the one with the highest *information gain*. A straightforward implementation of this step is as follows: for each possible pattern, we compute the new $u^*(t)$ via iterative scaling assuming the pattern has been added to T and we calculate the new $D_{KL}(v \| u^*)$; line 5 then adds to T the pattern that gives the greatest improvement in D_{KL}. Line 6 updates $u^*(t)$ based on the pattern that ended up being added to T in this iteration of the loop, i.e., it runs iterative scaling to compute new $\lambda(t)$ values for the existing patterns and the new pattern, as described in Section 2.1, and computes a new $u^*(t)$ for each tuple in D as per Equation 2. Note that rather than adding patterns to T until the KL-divergence threshold is satisfied, the algorithm can be easily modified to terminate after adding a specified number of patterns to T.

A more efficient but inexact implementation of line 4 is to *estimate* the information gain of a pattern p only based on the difference between the old and new u^* estimates for tuples in p's support set (as was also done in [10, 11]), not the difference between the old and new estimates for all of D (as we did by comparing $D_{KL}(v \| u^*)$ with and without p). Rather than running iterative scaling for each candidate pattern to compute a new $D_{KL}(v \| u^*)$, we compute an estimated information gain as follows. Let $g_{D,u^*}(p) = \frac{1}{|S_D(p)|} \sum_{t \asymp p} u^*(t)$. This is the estimated fraction of tuples with outcome 1 in p's support set based on the maximum-entropy estimate implied by the patterns currently in T. We compare this with the actual fraction of tuples with outcome 1 in p's support set, which is $f_{D,v}(p)$. The larger the difference, weighted by the size of p's support set, the more beneficial it is to add p to the explanation table so that the outcomes of tuples in p's support set can be estimated more precisely. Formally, we define $gain(p)$ as follows, dropping the subscripts of $f(p)$ and $g(p)$ when it is clear what they are defined over.

Query 1 Generating all candidate patterns and their gain estimates

```
SELECT (D.$A_1$,..., D.$A_d$) AS $p$,
       gain(count(*), AVG(D.$v$),AVG(D.$u^*$))
FROM D
CUBE BY D.$A_1$,..., D.$A_d$
ORDER BY gain DESC
LIMIT 1
```

$$|S_D(p)| \cdot \left(f(p) \log \left(\frac{f(p)}{g(p)} \right) + (1 - f(p)) \log \left(\frac{1 - f(p)}{1 - g(p)} \right) \right) \quad (3)$$

Query 1 shows an SQL implementation of line 4, assuming a user-defined function `gain` to compute the gain formula from Equation 3, and assuming that the u^*'s are maintained in an additional column of D. For each pattern p, the gain function requires count(*), which is $|S_D(p)|$, the average value of v, which is $f(p)$, and the average value of u^*, which is $g(p)$. Note the use of the CUBE BY operator to compute the gain of every pattern in $\mathcal{P}(D)$, and the ORDER BY and LIMIT clauses to select one pattern with the highest gain.

Unfortunately, despite using Equation 3 to approximate gain instead of running iterative scaling and computing a new $D_{KL}(v \| u^*)$ for every candidate pattern, Query 1 can still be prohibitively expensive. This is because the number of possible patterns can be very large: as many as $2^d \times |D|$. In particular, it ran for over three hours on one of the data sets that we will use in Section 5, which has 1.5 million tuples, 9 columns and 78 million patterns. Furthermore, the gain formula from Equation 3 is not amenable to existing optimizations for computing aggregates over data cubes (e.g., [2, 15]). Another source of complexity is that greedy explanation table construction is not *anti-monotonic*, meaning that a pattern that has low gain in one iteration of the while loop in Algorithm 1 cannot be pruned from consideration because its gain might become higher in subsequent iterations as more patterns are added to T. For example, consider the following patterns from an Exercise relation similar to that in Table 1: p_1=(*,*,*), p_2=(Sat,*,*) and p_3=(Sat, Afternoon,*). Note that p_1 contains p_2 which contains p_3. Suppose p_1 and p_3 both contain exactly the same fraction of tuples with $v = 1$, but the fraction of tuples with $v = 1$ covered by p_2 is very different. If p_1 happens to be added to the explanation table first, then the information gain from p_3 is zero. However, if p_2 is added next, then p_3 will have a non-zero gain because the maximum-entropy estimates u^* will be based on the assumption that all Saturday tuples, including the Saturday afternoon ones covered by p_3, have a similar distribution of the outcome attribute v.

We refer to Algorithm 1 as *Baseline*. Note that line 4 is the bottleneck as it needs to compute gain for a very large number of patterns whenever a new pattern needs to be added to T; line 6 is relatively inexpensive as it only needs to run iterative scaling once, after the best pattern has been selected. Since we cannot prune patterns from consideration throughout the execution of Algorithm 1, we propose to improve performance (at the expense of accuracy) using sampling.

4.2 A Straightforward Sampling Approach

A straightforward sampling approach to constructing explanation tables is to draw a random sample s from D and run Algorithm 1 on s instead of D. A useful strategy is to draw a new random sample s before starting the next iteration (the cost of re-sampling at each iteration is negligible compared to other costs, and re-sampling can compensate for the rare case of a bad sample). The set of possible patterns in line 4 becomes $\mathcal{P}(s)$, i.e., the data cube of s with respect to the attributes A_1 through A_d, which should be much smaller than $\mathcal{P}(D)$. The running time of Query 1, which now operates over s instead of D, becomes independent of the size of D. We call this straightforward sampling approach *SampleCube*.

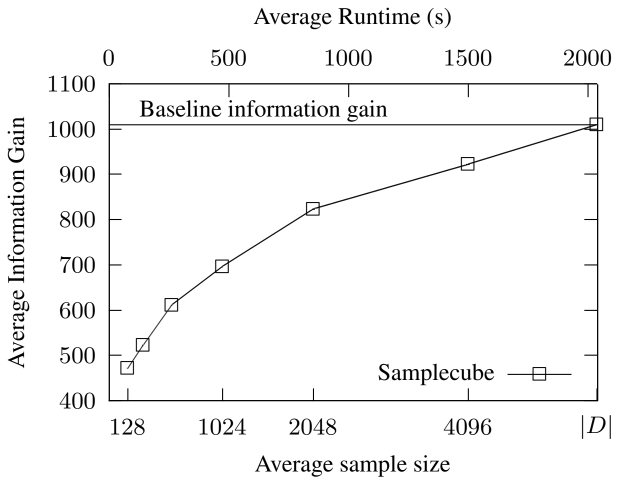

Figure 1: Average information gain and running time of *SampleCube* as a function of sample size (using `Upgrade`).

Instead of testing all possible patterns, *SampleCube* only considers those which appear in the data cube of the sample. The probability of a pattern belonging to $\mathcal{P}(s)$ and therefore being considered for inclusion in T is proportional to its support. For example, there are two Friday tuples and five Saturday tuples in Table 1; therefore, a Saturday tuple is more likely to be included in a random sample than a Friday tuple, and the pattern (Sat,*,*) is more likely to be considered than (Fri,*,*). This is a reasonable strategy since adding a pattern with high support into T can improve the u^* estimate for more tuples and therefore increase the information content of the explanation table.

The efficiency of *SampleCube* over *Baseline* comes at the cost of accuracy. Figure 1 quantifies the tradeoff between runtime and information content. We tested *Baseline* and *SampleCube* on a data set we refer to as `Upgrade`. It contains 5795 airline ticket records from United Airlines, recording various attributes about the tickets (origin airport, destination airport, flight date and time, booking class, etc.) and a binary outcome attribute indicating whether the passenger was upgraded to business class. We ran *SampleCube* five times, each time choosing a different random sample of a given size, and report the average numbers across the five runs. The sample size is shown on the lower x-axis and it varies from 128 to 5795. The upper x-axis shows the running time to create an explanation table with 50 patterns: *Baseline* took just over 2000 seconds whereas the running time of *SampleCube* increases as the sample size increases. Note that the upper x-axis is linear and was used to decide the positions of the numbers on the lower x-axis. The y-axis shows the average information gain of the constructed information tables. We define this as the improvement in D_{KL} compared to an explanation table with only the all-wildcards pattern. The horizontal line at an information gain of just over 1000 corresponds to *Baseline*.

As Figure 1 shows, *SampleCube* is not particularly effective: a very large sample size is required to approach the information gain

Query 2 Gain calculation in *SampleCube gain$_D$*

```
SELECT (P.$A_1$,..., P.$A_d$) AS $p$,
       gain(count(D.*), AVG($D.v$), AVG($D.u^*$))
FROM D, $\mathcal{P}(s)$ AS P
WHERE  (P.$A_1$ ISNULL OR P.$A_1$ = D.$A_1$) AND ...
       AND (P.$A_d$ ISNULL OR P.$A_d$ = D.$A_d$)
CUBE BY P.$A_1$,..., P.$A_d$
ORDER BY gain DESC
LIMIT 1
```

of *Baseline*. For example, to achieve an information gain of 900, which is 90 percent of *Baseline*'s information gain of 1000, we need a sample size of 4096, on which *SampleCube* ran for 1500 seconds, which is 75 percent of the time taken by *Baseline*. It might appear that the problem is with the smaller candidate pattern space of *SampleCube*: $\mathcal{P}(s)$ versus $\mathcal{P}(D)$. However, upon closer examination, we found that $\mathcal{P}(s)$ did in fact include the patterns that *Baseline* was adding to the explanation table. The best patterns were in the candidate set of *SampleCube* but they were not being selected because their gain was not high enough. The problem is that the information gain of these patterns was not being estimated accurately using the sample compared to using the entire relation.

To verify this claim, we implemented a new algorithm that is effectively a hybrid of *Baseline* and *SampleCube*. Before we explain the new algorithm, we define $gain_D(p)$ to be the result of Equation 3 using the actual fraction of tuples with 1-outcomes in p's support set, i.e., $f_{D,v}(p)$, and the estimates $g_{D,u^*}(p)$. This is the information gain of adding p to T in terms of being able to estimate $v(t)$ over D. In contrast, we define $gain_s(p)$ to be the result of Equation 3 using the f and g values computed from the sample s, i.e., $f_{s,v}(p)$ and $g_{s,u^*}(p)$. This is the information gain of adding p to T in terms of being able to estimate $v(t)$ over s. Note that *Baseline* computes $gain_D$ but *SampleCube* computes $gain_s$. The new algorithm considers the same candidate pattern space as *SampleCube*, namely $\mathcal{P}(s)$, but computes $gain_D$. We refer to the new algorithm as *SampleCube gain$_D$* and the original as *SampleCube gain$_s$*. Query 2 shows the SQL implementation of line 4 of *SampleCube gain$_D$*. Note the join of $\mathcal{P}(s)$ with D to obtain $gain_D$ and the use of `ISNULL` to compute support sets for patterns with wildcards.

Figure 2 shows the results using the `Upgrade` data set, with the sample size on the x-axis and the average information gain over five runs with different random samples on the y-axis; both algorithms were given exactly the same samples. The difference is dramatic: on this data set, *SampleCube gain$_D$* approaches the information gain of *Baseline* with a sample size as small as 16. This confirms that restricting the candidate pattern space to those which appear in the sample is an effective optimization, as long as we compute $gain_D$ instead of $gain_s$. Of course, computing $gain_D$ for patterns in $\mathcal{P}(s)$ (Query 2) requires a data cube over p joined with D, which is much more expensive than computing $gain_s$. In the next section, we show how to compute $gain_D$ for patterns in $\mathcal{P}(s)$ more efficiently.

4.3 The Flashlight Strategy

As we explained in the previous section, sampling can be used to obtain good explanation tables, but only when combined with accurate gain estimates obtained by taking the entire relation into account. In this section, we introduce the *Flashlight* strategy that produces the same output as Query 2, but instead of joining inputs of sizes $|\mathcal{P}(s)|$ and $|D|$, it only needs to join inputs of sizes $|s|$ and $|D|$. Since $\mathcal{P}(s)$ can be much larger than s, *Flashlight* significantly improves the performance of explanation table construction, as we will show experimentally in Section 5.

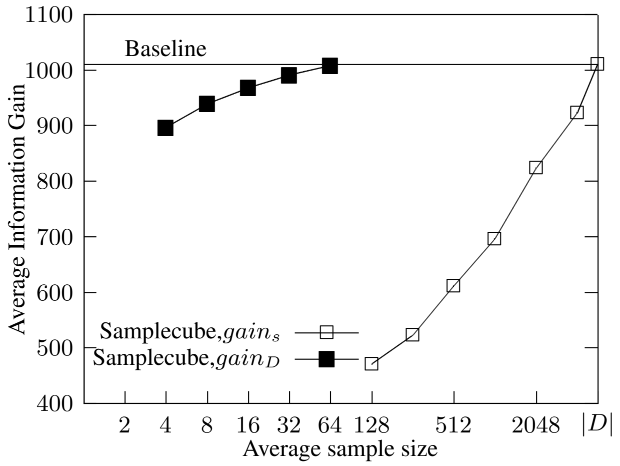

Figure 2: Average information gain of two *SampleCube* variants as a function of sample size (using `Upgrade`).

Table 5: A sample of three tuples from the Exercise table

day	time	meal	goal met?
Sun	Morning	Banana	Yes
Thu	Dawn	Oatmeal	Yes
Sat	Dawn	Banana	No

First, we need the following definitions.

Definition 4. Let $p = lcap(t_1, t_2)$ be the lowest common ancestor pattern of tuples t_1 and t_2. Then $p[A_j] = t_1[A_j]$ when $t_1[A_j] = t_2[A_j]$ and $p[A_j] =$ '*' otherwise.

For example, $lcap$((Fri,Dawn,Banana), (Fri,Night,Green salad) is (Fri,*,*) and $lcap$((Fri,Dawn,Banana), (Sat,Midday,Toast)) is (*,*,*). Intuitively, $lcap(t_1, t_2)$ gives the "tightest" or most specific pattern that matches both tuples.

Definition 5. The lowest common ancestor cross-product, $LCA(D_1, D_2)$, of two relation instances, D_1 and D_2, is $\bigcup_{t_i \in D_1} \bigcup_{t_j \in D_2} lcap(t_i, t_j)$.

Table 6 shows the LCA of the Exercise relation from Table 1 with a sample of three tuples from it, as shown in Table 5. Each row of Table 6 corresponds to one of the 14 tuples from the Exercise table and each column to one of the three tuples from the sample. Note that Table 6 assumes a duplicate-preserving version of LCA, meaning that the same lowest common ancestor pattern can be derived from multiple combinations of a tuple from Exercise and a tuple from the sample. For example, the underlined pattern (*,*,Banana) appears in the LCA five times.

4.3.1 Algorithm Description

The insight behind *Flashlight* is that $LCA(s, D)$ can be used to compute $gain_D$ without computing the costly cross-product of $\mathcal{P}(s)$ with D that is done in Query 2. We explain *Flashlight* using the following worked example. Take the Exercise relation from Table 1 and the sample from Table 5. Assume the (*,*,*) pattern

Table 6: Computing the LCA of the Exercise table with the sample from Table 5

`relation \Sample`	(Sun,Morning, Banana)	(Thu,Dawn,Oatmeal)	(Sat,Dawn, Banana)
(Fri,Dawn ,Banana)	(*,*,Banana)	(*,Dawn ,*)	(*,Dawn ,Banana)
(Fri,Night ,Green salad)	(*,*,*)	(*,*,*)	(*,*,*)
(Sun,Dusk ,Oatmeal)	(Sun,*,*)	(*,*,Oatmeal)	(*,*,*)
(Sun,Morning ,Banana)	(Sun,Morning,Banana)	(*,*,*)	(*,*,Banana)
(Mon,Afternoon ,Oatmeal)	(*,*,*)	(*,*,Oatmeal)	(*,*,*)
(Mon,Midday ,Banana)	(*,*,Banana)	(*,*,*)	(*,*,Banana)
(Tue,Morning ,Green salad)	(*,Morning,*)	(*,*,*)	(*,*,*)
(Wed,Night ,Burgers)	(*,*,*)	(*,*,*)	(*,*,*)
(Thu,Dawn ,Oatmeal)	(*,*,*)	(Thu ,Dawn ,Oatmeal)	(*,Dawn ,*)
(Sat,Afternoon ,Nuts)	(*,*,*)	(*,*,*)	(Sat ,*,*)
(Sat,Dawn ,Banana)	(*,*,Banana)	(*,Dawn ,*)	(Sat ,Dawn ,Banana)
(Sat,Dawn ,Oatmeal)	(*,*,*)	(*,Dawn ,Oatmeal)	(Sat ,Dawn ,*)
(Sat,Dusk ,Rice)	(*,*,*)	(*,*,*)	(Sat ,*,*)
(Sat,Midday ,Toast)	(*,*,*)	(*,*,*)	(Sat ,*,*)

has just been added to T, meaning that every tuple in D now has a u^* of 0.5; recall that we assumed an additional column of D to store the u^*'s. We now show how *Flashlight* implements line 4 of the greedy algorithm for selecting the next pattern for T, i.e., computing $gain_D$ of all the tuples in $\mathcal{P}(s)$ and choosing the one with the highest gain.

The first step is to compute the cross-product $LCA(s, D)$, followed by a group-by aggregation on each unique pattern that computes the following: $count(D.*)$, $SUM(D.v)$ and $SUM(D.u^*)$. Table 7 shows these aggregates for the 13 distinct patterns in $LCA(s, D)$; we will explain the "All ancestors" column shortly. Take the (*,*,Banana) pattern for instance. It appears five times in the LCA set, so its count is five. To understand $SUM(v)$, note that the five occurrences of (*,*,Banana) in the LCA set are derived from (Fri,Dawn,Banana) which has $v = 1$, (Sun,Morning,Banana) which has $v = 1$, (Mon,Midday,Banana) twice which has $v = 1$, and finally (Sat,Dawn,Banana) which has $v = 0$. The sum of v's is four, including counting (Mon,Midday,Banana) twice. Similarly, the sum of u^*'s is 2.5, including counting (Mon,Midday,Banana) twice; recall that at this point every tuple in D has $u^* = 0.5$.

An important observation is that at this point, the above aggregates do not correspond to the support or the $f(p)$ and $g(p)$ values of the patterns. For instance, the support of (*,*,Banana) in D is four tuples, but its count in the LCA is five.

In the second step, for each pattern p produced as output of the previous step, we generate its *ancestor patterns*, defined as p itself plus all other patterns that are the same as p except one or more of the attribute values in p have been replaced by wildcards. These are shown in the rightmost column of Table 7. For each ancestor pattern of p, we associate with it exactly the same count, $SUM(v)$ and $SUM(u^*)$ values as p. We then take the ancestor patterns of all patterns in the LCA and compute another group-by aggregation on each unique pattern to get a new count, $SUM(v)$ and $SUM(u^*)$. These are shown in the four leftmost columns of Table 8. For example, (*,*,Banana) is an ancestor of itself (with count 5), of (*,Dawn,Banana) (with count 1), of (Sat,Dawn,Banana) (with count 1) and of (Sun,Morning,Banana) (with count 1). Thus, its count in Table 8 is eight, and $SUM(v)$ and $SUM(u^*)$ are computed similarly.

Note that at this point, after generating the ancestor patterns of those in the LCA, the resulting set of 20 patterns is exactly the set $\mathcal{P}(s)$, i.e., the data cube of s.

The third step is to correct the count and sum aggregates so they correspond to $|S_D(p)|$, $f_{D,v}(p)$ and $s_{D,u^*}(p)$. The trick is to realize that we do not have to probe D again, but rather we only need to compare what we have computed so far with the sample s. For example (*,*,Banana) matches two tuples in s (shown in Table 5), so it suffices to divide the aggregates we have computed by two. This gives the corrected $count = 4$, $SUM(v) = 3$ and $SUM(u^*) = 2$. From these, we compute $f_{D,v}$(*,*,Banana)$=\frac{SUM(v)}{count} = \frac{3}{4}$ and g_{D,u^*}(*,*,Banana)$=\frac{SUM(u^*)}{count} = \frac{2}{4}$. The rightmost four columns of Table 8 show the "frequency in sample" of each candidate pattern p (i.e., how many tuples in s it matches), the corrected count (i.e., $|S_D(p)|$), and $f_{D,v}(p)$ and $g_{D,u^*}(p)$.

The final step is to select the pattern from $\mathcal{P}(s)$ with the highest $gain_D$. We have all the aggregates we need for this, and we evaluate Equation 3 for each candidate pattern. In this round, the pattern (Sat,*,*) has the highest gain of $5 * (0 + \log(\frac{1}{0.5})) = 5\log(2)$, so it is added to the explanation table.

4.3.2 Complexity Analysis

We now discuss the computational complexity of *Flashlight*, which is $O(|s||D| + |\mathcal{P}(s)||s|)$. Recall that Query 2 requires $|\mathcal{P}(s)||D|$ operations to compute $gain_D$ for patterns in $\mathcal{P}(s)$. On the other hand, the first step of *Flashlight* only requires $|s||D|$ operations to compute $LCA(s, D)$, the second step needs $|\mathcal{P}(s)|$ operations to generate ancestor patterns, and the third step needs $|\mathcal{P}(s)||s|$ operations to join ancestor patterns with the sample to compute the corrected aggregates.

4.3.3 SQL Implementation

We implemented *Flashlight* in PostgreSQL as a series of non-materialized SQL views that include joins and group-by aggregation, and call two user-defined functions: one for computing gain (Equation 3) and one for generating the ancestor patterns of a given pattern p. The first view computes $LCA(s, D)$ and a group-by aggregation query over it. For each such pattern, the second view generates its ancestors and computes a group-by aggregation query to obtain the required counts and averages (recall Table 8). The third view joins all the patterns in $\mathcal{P}(s)$ with s to compute the corrected aggregates.

One advantage of our SQL implementation is that we can leave it to the query optimizer to select the best plan, including suitable join and aggregation techniques depending on the characteristics of the data. In our experiments, Postgres chose a nested-loops join for $LCA(s, D)$ and hash-aggregate for the group-bys.

4.4 The Laserlight Strategy

The *Flashlight* strategy from the previous section works with a reduced space of candidate patterns ($\mathcal{P}(s)$ instead of $\mathcal{P}(D)$), but

Table 7: Computing aggregates over the LCA from Table 6 and generating all ancestor patterns

Pattern	`count`	`SUM`(v)	`SUM`(u^*)		`All ancestors`
(*,*,*)	21	9	10.5	$\rightarrow$	(*,*,*)
(*,*,Banana)	5	4	2.5	$\rightarrow$	(*,*,Banana) ,(*,*,*)
(*,Dawn ,*)	3	2	1.5	$\rightarrow$	(*,Dawn ,*) ,(*,*,*)
(Sat ,*,*)	3	0	1.5	$\rightarrow$	(Sat ,*,*) ,(*,*,*)
(*,*,Oatmeal)	2	2	1	$\rightarrow$	(*,*,Oatmeal), (*,*,*)
(*,Dawn ,Banana)	1	0	.5	$\rightarrow$	(*,Dawn ,Banana),(*,Dawn,*),(*,*,Banana) ,(*,*,*)
(*,Dawn ,Oatmeal)	1	0	.5	$\rightarrow$	(*,Dawn ,Oatmeal),(*,Dawn,*),(*,*,Oatmeal),(*,*,*)
(*,Morning,*)	1	0	.5	$\rightarrow$	(*,Morning,*),(*,*,*)
(Sat ,Dawn ,*)	1	0	.5	$\rightarrow$	(Sat,Dawn,*),(Sat ,*,*),(*,Dawn ,*),(*,*,*)
(Sat ,Dawn ,Banana)	1	0	.5	$\rightarrow$	(Sat,Dawn,Banana),. . . , (Sat ,*,*),(*,Dawn ,*),(*,*,*)
(Sun,*,*)	1	1	.5	$\rightarrow$	(Sun,*,*),(*,*,*)
(Sun,Morning,Banana)	1	1	.5	$\rightarrow$	(Sun,Morning,Banana),. . . , (*,Morning,*),(*,*,Banana),(*,*,*)
(Thu ,Dawn ,Oatmeal)	1	1	.5	$\rightarrow$	(Thu,Dawn,Oatmeal),. . . ,(*,Dawn,*),(*,*,Oatmeal),(*,*,*)

Table 8: Computing aggregates over $\mathcal{P}(s)$

Pattern	`count`	`SUM`(v)	`SUM`(u^*)	`Freq. in sample`	$\|S_D(p)\|$	$f_{D,v}(p)$	$g_{D,u^*}(p)$
(*,*,*)	42	21	21	3	14	.5	.5
(*,*,Banana)	8	6	4	2	4	.75	.5
(*,*,Oatmeal)	4	3	2	1	4	.75	.5
(*,Dawn ,*)	8	4	4	2	4	.5	.5
(*,Morning,*)	2	1	1	1	2	.5	.5
(Sat ,*,*)	5	0	2.5	1	5	0	.5
(Sun,*,*)	2	2	1	1	2	1	.5
(Thu,*,*)	1	1	.5	1	1	1	.5
(*,Dawn,Banana)	2	1	1	1	2	.5	.5
(*,Dawn ,Oatmeal)	2	1	1	1	2	.5	.5
(*,Morning,Banana)	1	1	.5	1	1	1	.5
(Sat,*,Banana)	1	0	.5	1	1	0	.5
(Sun,*,Banana)	1	1	.5	1	1	1	.5
(Thu,*,Oatmeal)	1	1	.5	1	1	1	.5
(Sat,Dawn,*)	2	0	1	1	2	0	.5
(Sun,Morning,*)	1	1	.5	1	1	1	.5
(Thu,Dawn,*)	1	1	.5	1	1	1	.5
(Sun,Morning,Banana)	1	1	.5	1	1	1	.5
(Sat,Dawn,Banana)	1	0	.5	1	1	0	.5
(Thu,Dawn,Oatmeal)	1	1	.5	1	1	1	.5

computes $gain_D$, not $gain_s$. However, it still requires scanning D at each iteration to compute $LCA(s, D)$. In contrast, our second strategy, *Laserlight*, improves *SampleCube* by optimizing its running time rather than its accuracy, computing $gain_s$ rather than $gain_D$, but for a smaller set of patterns. As a result, it enables larger sample sizes than *SampleCube*.

We start with the following observation. Recall Table 8, which lists all 20 patterns in $\mathcal{P}(s)$ using s from Table 5, with statistics computed based on D from Table 1. Consider the following two patterns from $\mathcal{P}(s)$: p_1=(Sat,*,Banana) and p_2=(Sat,Dawn,Banana). They have exactly the same support set over D as they both match only tuple 11. As a result, they will have the same gain in every iteration of the explanation table construction algorithm. Similarly, two patterns that overlap significantly will have very similar gain. The idea behind *Laserlight* is to avoid considering redundant patterns, which are common in real data sets due to correlations, and therefore speed up the process of choosing the pattern with the highest gain.

To accomplish this, we revisit the idea of lowest common ancestor patterns, which we used in *Flashlight* to efficiently compute $gain_D$ for patterns in $\mathcal{P}(s)$. The insight is that the pattern set $LCA(s, s)$ does not contain any redundant patterns, as each such pattern is the lowest descendant for which a unique pair of tuples co-occurs in the same support set and, hence, mutually non-redundant. This is exactly the new *Laserlight* strategy: we generate a random sample s and run Algorithm 1 on the sample, as in *SampleCube $gain_s$*, but with line 4 choosing the best pattern only from those in $LCA(s, s)$.

To quantify how many patterns in $\mathcal{P}(s)$ are redundant, Figure 3 plots three quantities as functions of the sample size over the `Upgrade` data set, calculated as averages over five runs with different uniform samples; note that both axes are logarithmic. The first is $|\mathcal{P}(s)|$, which is 7229 for $|s| = 64$ and grows to approximately 350,000, which is $|\mathcal{P}(D)|$. The other two lines are very similar: the number of patterns with unique support sets in $\mathcal{P}(s)$, labeled "$|unique\_supp_s(\mathcal{P}(s))|$" and the size of $LCA(s, s)$. At $|s| = 64$, there are only roughly 330 of these, meaning that most patterns in $\mathcal{P}(s)$ are in fact redundant in this example. This means that *Laserlight* can be significantly more efficient in practice than *SampleCube $gain_s$*.

We now give a worked example of *Laserlight* using the same input as in the worked example of *Flashlight* in the previous section: D is the Exercise relation from Table 1, s consists of the three tuples from Table 5; furthermore, assume the (*,*,*) pattern has just

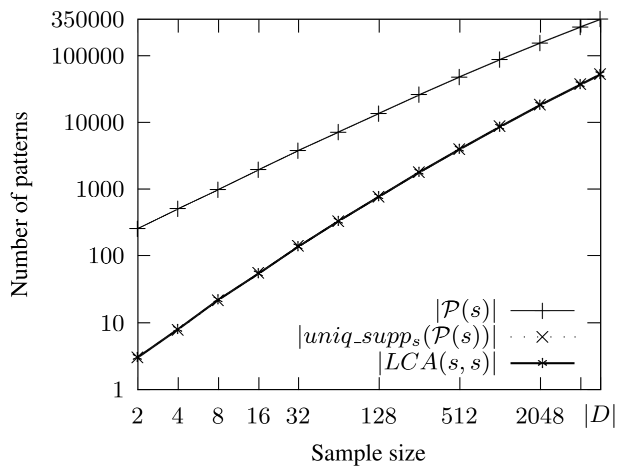

Figure 3: Number of patterns as a function of sample size using `Upgrade`

Table 9: Computing aggregates over $LCA(s,s)$

Pattern	$\|S_s(p)\|$	$f_{s,v}(p)$	$g_{s,u^*}(p)$
(*,*,*)	3	$\frac{2}{3}$	$\frac{2}{3}$
(*,*,Banana)	2	$\frac{1}{2}$	$\frac{2}{3}$
(*,Dawn,*)	2	$\frac{1}{2}$	$\frac{2}{3}$
(Sun,Morning,Banana)	1	1	$\frac{2}{3}$
(Thu,Dawn,Oatmeal)	1	1	$\frac{2}{3}$
(Sat,Dawn,Banana)	1	0	$\frac{2}{3}$

been added to T, meaning that every tuple in s now has a u^* of $\frac{2}{3}$, which is the fraction of 1-outcomes in the sample (not in D, as was the case for *Flashlight*). Assume the table storing the sample s includes an additional column to store the u^*'s.

To select the next pattern into T, the first step is to compute $LCA(s,s)$, which is shown in the leftmost column of Table 9. There are only six patterns in this set, as compared to 20 in $\mathcal{P}(s)$ (recall Table 8). Step two is to scan s and compute the following three aggregates for each pattern in $LCA(s,s)$: the support $|S_s(p)|$ and the fractions $f_{s,v}(p)$ and $g_{s,u^*}(p)$. These are shown in Table 9. The final step is to select the pattern with the highest $gain_s$, which turns out to be (Sat,Dawn,Banana). Its $gain_s$ is $0 \log 0 + 1 \log(1/\frac{1}{3}) = \log 3$. Recall from the previous section that *Flashlight* chose the pattern (Sat,*,*) next, whereas this pattern is not even in the candidate set of *Laserlight*.

4.4.1 Complexity Analysis

Computing the pattern set $LCA(s,s)$ takes $|s|^2$ operations and joining this set with the sample to compute $gain_s$ requires $|LCA(s,s)||s|$ operations. Since $\mathcal{P}(s)$ can be much larger than $LCA(s,s)$, as we showed earlier, *Laserlight* can be much more efficient than *SampleCube gain$_s$*, whose complexity is $|\mathcal{P}(s)||s|$.

4.5 Discussion

We conclude this section with a summary of the algorithms in Table 10; for completeness, we include the two *SampleCube* techniques which are stepping stones to *Flashlight* and *Laserlight*. The algorithms differ in the set of candidate patterns considered and the method for computing gain. Since $LCA(s,s) \subseteq \mathcal{P}(s) \subseteq \mathcal{P}(D)$,

Table 10: A comparison of algorithms for explanation table construction

Algorithm	Candidate pattern space	gain estimation
Baseline	$\mathcal{P}(D)$	$gain_D$
SampleCube gain$_s$	$\mathcal{P}(s)$	$gain_s$
SampleCube gain$_D$	$\mathcal{P}(s)$	$gain_D$
Flashlight	$\mathcal{P}(s)$	$gain_D$
Laserlight	$LCA(s,s)$	$gain_s$

Baseline is the most accurate (i.e., it should have the highest information gain), but least efficient. *Flashlight* should be much faster but less accurate, followed by *Laserlight*, which should be the fastest but likely the least accurate. *Flashlight* constructs the same explanation tables as *SampleCube gain$_D$* but is faster.

5. EXPERIMENTS

In this section, we present our experimental results. We compare the information content of explanation tables against existing methods including decision trees and database summaries (Section 5.2), and we compare the proposed explanation table construction algorithms in terms of running time, scalability and information content (Section 5.3).

5.1 Experimental Setup

The experiments were performed on a 3.3 GHz AMD machine with 16 GB of Ram and 6 MB of cache, running Ubuntu Linux. The proposed algorithms are implemented in PostgreSQL 9.1. No personally identifiable information was gathered or used in conducting this study. Our analysis is based on the following anonymous and/or aggregated data sets.

- `Upgrade`: 5795 United Airline ticket records obtained from `https://www.diditclear.com/`. Each record contains seven attributes about the tickets (origin airport, destination airport, date, booking class, etc.) and a binary attribute indicating whether the passenger was upgraded to business class. This data set has $|\mathcal{P}(D)| = 346,532$.

- `Adult`: U.S. Census data containing demographic attributes such as occupation, education and gender, and a binary outcome attribute denoting whether the given person's income exceeded \$50K. This data set was downloaded from the UCI archive at `archive.ics.uci.edu/ml/datasets/Adult/` and contains 32561 tuples, 8 attributes and $436,414$ possible patterns.

- `Income`: U.S. Census data containing demographic attributes such as the number of children and marital status, and a binary outcome attribute indicating whether the given person's income exceeded $100,000$ dollars. This data set was downloaded from IPUMS-USA at `https://usa.ipums.org/usa/data.shtml/` and contains roughly 1.5 million tuples, 9 attributes and 78 million possible patterns.

Throughout this section, we measure the information content of explanation tables and other summaries in terms of the *average information gain*. As we explained in Section 4.2, this is the improvement in D_{KL} compared to having only the all-wildcards pattern and knowing only the fraction of tuples in the whole data set having outcome 1.

5.2 Comparison of Explanation Tables with Related Approaches

We begin with a comparison of explanation tables with three related approaches: 1) decision trees, 2) the SURPRISE operator for data cube exploration [11], and 3) the summarization techniques from [5, 6] which find the fewest patterns that cover most of the tuples with outcome 1.

For decision trees, we used the Weka J48 implementation [7] of the C4.5 algorithm, which provides a *confidence* parameter to control pruning and the number of tuples a leaf node can cover. We consider both multi-way split decision trees in which a node can have many children (which we abbreviate DT) and binary split decision trees in which a node has exactly two children (which we abbreviate DTN). Since there is no direct way to enforce decision tree size in Weka, a common approach for finding the best decision tree parameters is to perform a grid search over the parameter space. In our experiments, we varied confidence in the range $[0.01 - 0.59]$ and the minimum number of tuples per leaf in the range $[2 - 20]$.

For the SURPRISE operator, which also uses information gain to find informative patterns but does not allow partially overlapping patterns, we tried different values of the accuracy threshold ϵ using `Upgrade` and `Adult`. A lower ϵ indicates a finer algorithm resolution but slower runtime. To fairly represent SURPISE, we have chosen two values for ϵ for each data set: SURPRISE_cmp, which uses ϵ that gives comparable runtime to *Flashlight*, and SURPRISE_bst, which uses ϵ that yields the best possible information gain given our CPU and memory resources.

The summaries from [5, 6] require a confidence threshold that denotes the minimum fraction of 1-outcomes per pattern. We experiment with a variety of confidence thresholds, including $\{0.4, 0.5, \ldots, 0.9, 0.95, 0.99\}$, and report the runtime of the one with the best information gain, denoted Cover_bst. After generating the summaries, we apply the maximum entropy principle to derive $u^*(t)$ for each tuple t in D and compute information gain.

Finally, to compute our explanation tables, we use *Flashlight* with a sample size of 16, which we observed to offer a good tradeoff between running time and information gain (recall Figure 2). The reported runtimes and information gains are averages of five runs with different samples.

Figure 4 plots the average information gain using `Upgrade` (left) and `Adult` (right) as a function of the number of patterns (or, in the case of decision trees, the number of leaves). DT and DTN are represented as individual points on the graphs, corresponding to the numbers of leaves we were able to obtain using the grid search explained above. *Flashlight* outperforms all other approaches except for small numbers of patterns (below 8) on the `Adult` data set, in which case SURPRISE has a slightly higher information gain. Upon further inspection, we observed that the problem is with our gain formula from Equation 3. The SURPRISE operator computes the exact KL-divergence for candidate patterns because doing so is feasible for non-overlapping patterns. One of the first patterns chosen by SURPRISE_bst gives a high improvement in KL-divergence, but a relatively modest gain, as calculated by Equation 3, and therefore is not chosen by *Flashlight*. We omit the results for our largest data set, `Income`, because many of the competing techniques ran out of memory.

Table 11 shows the corresponding running times, assuming each technique is required to generate 50 patterns. For DT and DTN, we report the total running time of the grid search since it is not possible to directly control decision tree size. "N/A" indicates that the algorithm ran out of memory and did not produce any output. We conclude that *Flashlight offers the best information gain, running time and scalability out of the tested algorithms.*

Table 11: Running time comparison

	`Upgrade`	`Adult`	`Income`
Flashlight	47 s	252 s	0.6 h
DT	1021 s	2599 s	56 h
DTN	6120 s	8348 s	N/A
SURPRISE_cmp	61 s ($\epsilon = 0.2$)	243 s ($\epsilon = 0.1$)	N/A
SURPRISE_bst	88 s ($\epsilon = 0.08$)	875 s ($\epsilon = 0.02$)	N/A
cover_bst	240 s ($\hat{c} = 0.99$)	2051 s ($\hat{c} = 0.6$)	N/A

5.3 Comparison of Explanation Table Construction Algorithms

5.3.1 Running Time

We start by showing the efficiency improvement of *Flashlight* as compared to *SampleCube gain*$_D$. Figure 5 shows the average running time (over five runs with different random samples) of *Flashlight* and a *Traditional plan* corresponding to the plan for Query 2 chosen by PostgreSQL, with the sample size on the x-axis. For comparison, the horizontal lines at the top of the figures show the running time for *Baseline*, which considers all possible patterns. The left figure refers to `Upgrade` with $|T| = 50$, the middle figure to `Adult` with $|T| = 50$ and the right figure to `Income` with $|T| = 20$ (we chose a smaller explanation table size for this large data set to keep the total running time of the very large number of experimental runs manageable). The performance advantage of *Flashlight* is dramatic: as the sample size increases, *Flashlight* becomes several orders of magnitude faster.

We also compared the running time of *Laserlight* and *SampleCube gain*$_s$, expecting the former to be much faster because its space of candidate patterns is smaller. We found that *Laserlight* was several times faster for sample sizes under 1000. For instance, using `Upgrade` and a sample size of 256, *Laserlight* took under 40 seconds while *SampleCube gain*$_s$ took 141 seconds.

5.3.2 Running Time vs. Information Gain

We now compare the "bang for the buck" of the proposed algorithms in terms of the best possible information gain they can provide for various running time budgets. To calculate the best possible information gain, we ran each algorithm with different sample sizes between 2 and 8192, and, for each sample size, we generated explanation tables of sizes between one and 50 patterns for `Upgrade` and `Adult` and between 1 and 20 for `Income`. Figure 6 plots the best average information gain (averaged over five runs with different samples) for various running time budgets. The left figure refers to `Upgrade`, the middle figure to `Adult` and the right figure to `Income`. Each point on the curves represents a combination of $|T|$ and $|s|$ that gave the best information gain at that running time budget for the given algorithm (labeled *Flashlight_bst*, *Laserlight_bst* and *SampleCube_bst*). We also separately plot a line for *Flashlight* with a sample size of 16, on which each point corresponds to a different number of patterns, to show that *Flashlight* performs well even at this small sample size. For context, we also draw horizontal lines to indicate the information gain of *Baseline*, even though its running time was extremely high and off the scale (as shown in Figure 5).

We make two observations from Figure 6. First, the information gain of *Flashlight* approaches that of *Baseline* for large running times (i.e., large $|T|$), but the running time of *Flashlight* is orders of magnitude less than that of *Baseline* (not shown in the figure). In fact, this is also true for a small sample of 16. Second, for sufficiently large running times, *Flashlight* offers the best

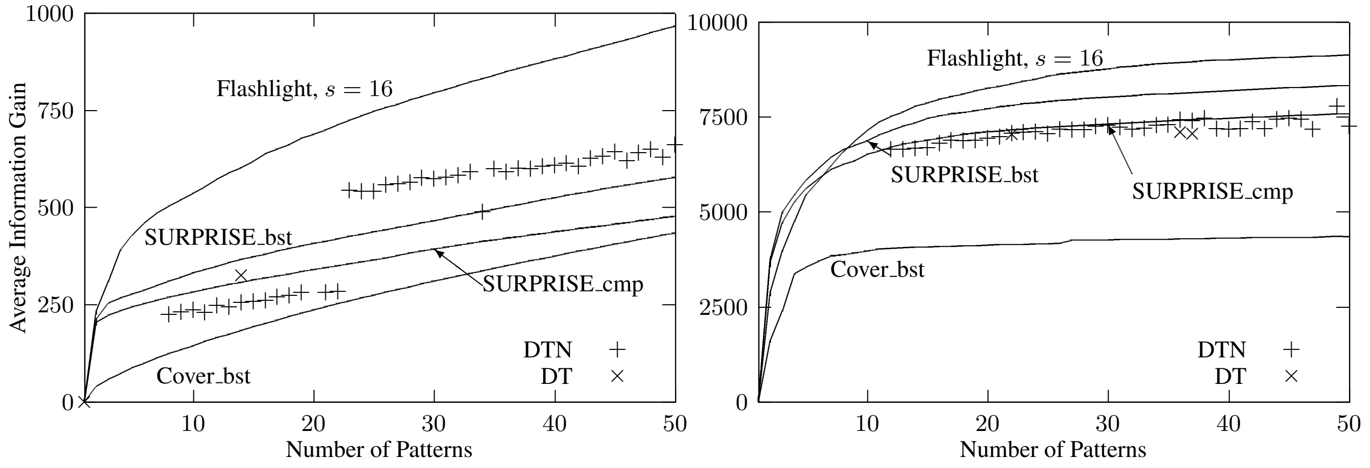

Figure 4: Average information gain of different techniques using `Upgrade` (left) and `Adult` (right)

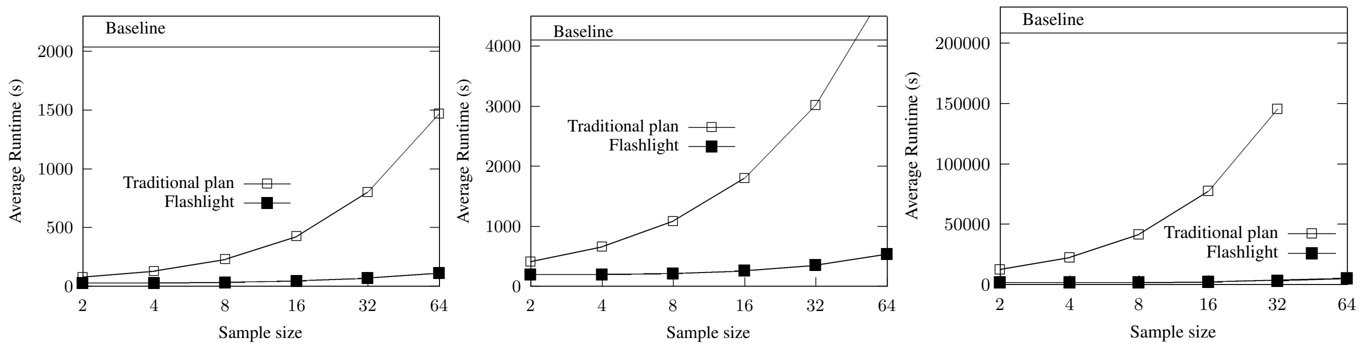

Figure 5: Running time of *Flashlight*, *SampleCube gain$_D$* and *Baseline* using `Upgrade` (50 patterns, left), `Adult` (50 patterns, middle) and `Income` (20 patterns, right)

tradeoff in running time versus information gain. Only for smaller running times (and not for every data set) does *Laserlight* becomes the method of choice. Specifically, *Laserlight* wins for runtimes under 28 seconds using `Adult` and runtimes under 619 seconds using `Income`. Further, *SampleCube gain$_s$* was never the preferred method in our experiments.

5.3.3 Scalability

Next, we investigate the scalability of *Flashlight* and *Laserlight* with respect to the data set size and the number of dimensions. To characterize the effect of $|D|$, we create smaller versions of our largest data set `Income` by sampling from it. We ensure a containment relationship among the samples of `Income`, e.g., all the tuples in the 50-percent sample are included in the 60-percent sample. Figure 7 shows the effect of varying the size of the portion of `Income` on *Flashlight* with $|s| = 16$ and $|T| = 20$ and *Laserlight* with $|s| = 512$ and $|T| = 20$. The running time of *Flashlight* grows linearly with $|D|$ since one of its steps needs to compute $LCA(s, D)$. On the other hand, the running time of *Laserlight* grows very slowly because it operates over s, not D. The reason why there is any growth in the running time at all is because our implementation of *Laserlight* re-samples from D after inserting a new pattern into the explanation table, and we need to scan D to generate a new sample.

In the final experiment, we vary the number of dimensions d. Again, we take `Income` and project out one dimension at a time. Results are shown in Figure 8. *Laserlight* is not affected by increasing the number of dimensions, but the running time of *Flashlight* increases roughly linearly. This is likely because $|\mathcal{P}(s)|$, which is the candidate pattern space of *Flashlight* grows more rapidly with d than $|LCA(s, s)|$, which is the candidate pattern space of *Laserlight*.

6. CONCLUSIONS

In this paper, we introduced the concept of explanation tables to summarize relations with binary outcome attributes in an interpretable and informative manner. We proved that constructing optimal explanation tables is NP-hard and we proposed novel heuristic algorithms for generating approximate solutions. Experimental results on real data sets showed the advantages of explanation tables over existing techniques in terms of information content and significant performance advantages of our explanation table construction algorithms over straightforward approaches.

We have two directions in mind for future work. The first is to make further advances in efficient and scalable construction of explanation tables by considering parallel and distributed evaluation, especially implementations using MapReduce. The second is to increase the complexity of the patterns included in explanation tables by adding negation and numeric attribute ranges.

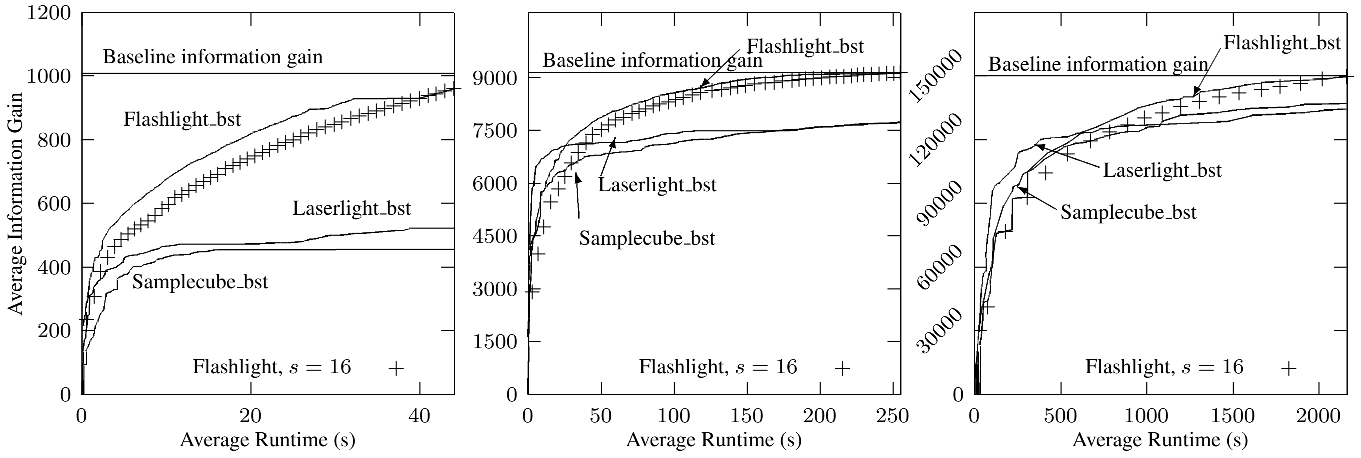

Figure 6: Information gain vs. running time for all algorithms using different parameter combinations using using `Upgrade` **(1-50 patterns, left),** `Adult` **(1-50 patterns, middle) and** `Income` **(1-20 patterns, right).**

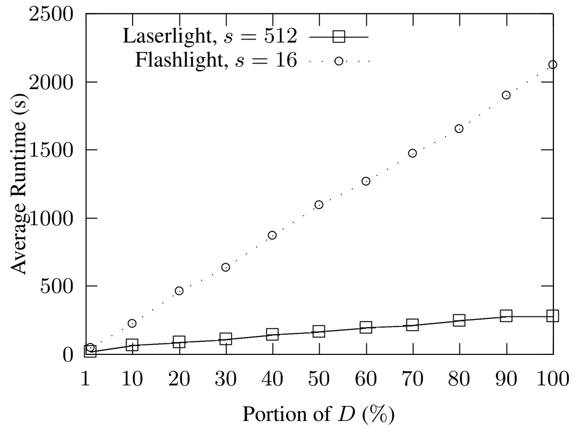

Figure 7: Scalability of *Flashlight* and *Laserlight* with $|D|$ using `Income` **(20 patterns).**

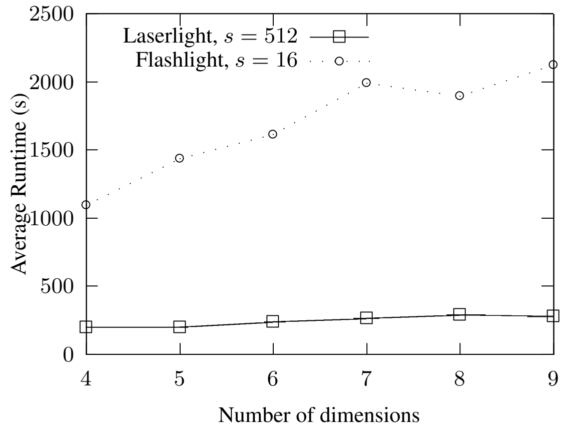

Figure 8: Scalability of *Flashlight* and *Laserlight* with $|d|$ using `Income` **(20 patterns).**

7. REFERENCES

[1] A. Berger, S. Della Pietra, V. J. Della Pietra: A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics* 22(1): 39-71 (1996)

[2] K. S. Beyer, R. Ramakrishnan: Bottom-Up Computation of Sparse and Iceberg CUBEs. *SIGMOD* 1999: 359-370

[3] S. Bu, L. V. S. Lakshmanan, R.T. Ng: MDL Summarization with Holes. *VLDB* 2005: 433-444

[4] F. Geerts, B. Goethals, T. Mielikainen: Tiling Databases. *Discovery Science* 2004, LNAI 3245: 278-289

[5] L. Golab, H. Karloff, F. Korn, D. Srivastava, B. Yu: On Generating Near-optimal Tableaux for Conditional Functional Dependencies. *PVLDB* 1(1): 376-390 (2008)

[6] L. Golab, F. Korn, D. Srivastava: Efficient and Effective Analysis of Data Quality using Pattern Tableaux. *IEEE Data Eng. Bull.* 34(3): 26-33 (2011)

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten: The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1): 10-18 (2009)

[8] K. N. Kontonasios, T. De Bie: An Information-Theoretic Approach to Finding Informative Noisy Tiles in Binary Databases. *SDM* 2010: 153-164

[9] L. V. S. Lakshmanan, R. Ng, C. Wang, X. Zhou, T. Johnson: The Generalized MDL Approach for Summarization. *VLDB* 2002: 766-777

[10] M. Mampaey, N. Tatti, J. Vreeken: Tell Me What I Need to Know: Succinctly Summarizing Data with Itemsets. *KDD* 2011: 573-581

[11] S. Sarawagi: User-Cognizant Multidimensional Analysis. *VLDB Journal* 10(2-3): 224-239 (2001)

[12] I. Witten, E. Frank, M. Hall: Data Mining: Practical Machine Learning Tools and Techniques, Morgan-Kaufmann 2011.

[13] E. Wu, S. Madden: Scorpion: Explaining Away Outliers in Aggregate Queries. *PVLDB* 6(8): 553-564 (2013)

[14] Y. Xiang, R. Jin, D. Fuhry, F. Dragan: Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. *KDD* 2008: 758-766

[15] D. Xin, J. Han, X. Li, B. W. Wah: Star-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration. *VLDB* 2003: 476-487