Exploring Data using Patterns: A Survey

Vargha Dadvar[a], Lukasz Golab[a], Divesh Srivastava[b]

[a] *University of Waterloo, 200 University Avenue, Waterloo, Ontario, N2L3G1, Canada*
[b] *AT&T Chief Data Office, 1 AT&T Way, Bedminster, NJ, 07921, USA*

Abstract

We present a survey of data exploration methods that extract multidimensional patterns from datasets consisting of dimension and measure attributes. These patterns are designed to summarize common properties of tuples sharing the same values of the measure attributes. We review motivating applications, we provide a categorization of the characteristics of patterns produced by various solutions to this problem, we categorize and experimentally evaluate commonly used performance optimizations, and we suggest directions for future research.

Keywords: Data exploration, Data summarization, Data explanation, Data cube, Pattern mining

1. Introduction

Data volumes have been growing rapidly in recent years. As a result, data-intensive methods are now common in many contexts, including business, science, and public governance. This motivates the need for tools that allow users who are not necessarily data management experts to explore large datasets. Such tools range from data visualization and aggregation [14, 19] to flexible search interfaces such as keyword search in structured databases [26].

In this paper, we focus on the exploration of datasets containing dimension attributes and binary or numeric measure attributes. In traditional business datasets, dimension attributes often describe products or employees, and measure attributes indicate sales totals or salaries. In Internet-of-Things (IoT) and infrastructure monitoring, dimension attributes may describe device properties and measure attributes correspond to performance statistics.

In Web datasets, dimension attributes may describe products, with aggregate user ratings as measure attributes. Additionally, in any of these applications, derived measure attributes may exist, e.g., a binary attribute denoting whether a given record was determined to be an outlier or to contain an error.

The *data cube* [12] has traditionally been used to explore these kinds of datasets, by allowing users to aggregate, roll-up and drill-down using various subsets of group-by attributes. However, in large-scale databases, the data cube may be very large and may not immediately reveal interesting trends. As argued in a recent vision paper by Vassiliadis and Marcel [22], next-generation Business Intelligence (BI) tools require new concepts and operators to help users discover information, among them those for automatic mining of models and patterns. This motivates the need for richer data exploration tools that can operate over data cubes and other multi-dimensional data models.

We observe that recent work on exploring multi-dimensional and OLAP datasets proposed a variety of methods to identify interesting fragments of the data, described using combinations of values of the dimension attributes [1, 2, 5, 7, 9, 11, 15, 18, 20, 21, 24, 25]. These value combinations are referred to as patterns. Below, we give examples to show that this class of methods provides *interpretable* summaries and explanations of trends in the data, aligning well with the anticipated needs of next-generation BI and data exploration tools.

Table 1 shows a flight dataset that will serve as a running example. For each flight, the dataset includes a record id, followed by three dimension attributes, Day of the week, flight Origin and flight Destination, as well as two measure attributes, a numeric attribute denoting how late the flight was and a binary attribute denoting whether the flight was full. First, note that the following two patterns summarize most of the tuples corresponding to full flights: (Day=*, Origin=*, Dest=London) and (Day=*, Origin=*, Dest=Frankfurt), where a star matches all the values of the corresponding attribute. In other words, full flights are mainly those that arrive in London or Frankfurt. Next, consider the pattern (Day=Mon, Origin=*, Dest=*), corresponding to flights scheduled on Mondays. This pattern may be interesting because none of these flights are full, which differs significantly from the fraction of full flights in the entire table. Finally, suppose a data analyst is surprised by the high average delay of flights in Table 1. Here, the pattern (Day=*, Origin=*, Dest=London) can serve as a potential explanation since

Table 1: A flight dataset

id	Day	Origin	Dest.	Late	Full
1	Fri	SF	London	20	1
2	Fri	London	LA	16	1
3	Sun	Tokyo	Frankfurt	10	1
4	Sun	Chicago	London	15	1
5	Sat	Beijing	Frankfurt	13	1
6	Sat	Frankfurt	London	19	1
7	Tue	Chicago	LA	5	0
8	Wed	London	Chicago	6	0
9	Thu	SF	Frankfurt	15	1
10	Mon	Beijing	SF	4	0
11	Mon	SF	London	7	0
12	Mon	SF	Frankfurt	5	0
13	Mon	Tokyo	Beijing	6	0
14	Mon	Frankfurt	Tokyo	4	0

flights arriving in London have some of the longest delays in the table.

We survey methods that automatically identify such patterns. Users may then inspect the patterns and explore tuples covered by the patterns. They may then extract patterns corresponding to smaller subsets of the data found to be interesting in the earlier exploration step, and so on. Of course, there are other approaches to data exploration and analysis, such as visualization and fitting models to predict the measure attributes based on the dimension attributes (we will comment on the role of prediction models in data cube exploration in Section 3.2). We restrict the scope of this survey to pattern-based exploration and refer the reader to [19] for a survey of data visualization methods. Furthermore, visualization may be thought of as complementary to pattern-based exploration. For example, interesting patterns may be identified, followed by visualizing the statistical properties of the corresponding tuples.

We make the following contributions towards an understanding of pattern-based data exploration methods.

1. We survey recent work on data exploration using multi-dimensional patterns and propose a categorization based on the properties of patterns suggested for exploration: *coverage*, *contrast*, and *information*.

2. We categorize and experimentally evaluate performance optimizations frequently used in pattern-based data exploration: *top-down pruning*, *row pruning*, *column pruning* and *parallel processing*. Notably, we experimentally demonstrate that some performance optimizations originally proposed for one method are also effective when applied to other methods. This observation should be of interest to researchers working in this area since it points out existing performance optimizations that may be beneficial for newly developed data exploration techniques.
3. We suggest open problems for future research.

In the remainder of this paper, we present the required background in Section 2, we categorize existing solutions in Section 3, we classify and experimentally evaluate performance optimizations in Sections 4 and 5, respectively, and we conclude in Section 6 with directions for future work.

A short version of this survey was presented at DOLAP 2021 [10]. This extended version includes two new sections with a classification (Section 4) and an experimental study (Section 5) of performance optimizations.

2. Background

We are given a dataset S with a set D of dimension attributes and a set M of measure attributes (also referred to as outcomes in some prior work [6]). Let $D_1, D_2, \ldots, D_d$ be the d dimension attributes and let $M_1, M_2, \ldots M_m$ be the m measure attributes. For now, we assume, as in the majority of previous work, that the dimension attributes are categorical, and we will comment on ordinal and numeric dimension attributes in Section 6. Measure attributes may be binary or numeric.

Let $dom(D_i)$ be the active domain of the ith dimension attribute. A pattern p is a tuple from $dom(D_1) \cup \{*\} \times \cdots \times dom(D_d) \cup \{*\}$, i.e., from the data cube over the dimension attributes, with '*' denoting all possible values of that attribute. A tuple $t \in S$ matches p, denoted by $t \asymp p$, if $p[D_j] =$ '*' or $t[D_j] = p[D_j]$ for each dimension attribute D_j. For example, tuple 4 from Table 1 matches the patterns $(*, *, *)$ and $(*, *, London)$, but does not match the pattern $(Fri, *, *)$; to simplify the notation, we drop attribute names from patterns and only include attribute values. Some approaches (e.g., [11]) support richer patterns with disjunctions and dimension hierarchies. Other methods described in this paper can also support disjunctions and dimension hierarchies, at the cost of a larger search space of candidate patterns. However, for simplicity of presentation, in the remainder of this paper, we will illustrate the methods using simple conjunctive patterns without hierarchies.

Let $sup(p)$ be the support of p in S, i.e., the number of tuples matching p, and let $sup_r(p)$ be the number of tuples matching p and satisfying a predicate r over the measure attributes. For example, $sup(*, *, London) = 4$ and $sup_{Full=0}(*, *, London) = 1$. Furthermore, let $\theta_r(p) = \frac{sup_r(p)}{sup(p)}$, which is the fraction of tuples matching p that also satisfy the predicate r. For example, $\theta_{Full=1}(*, *, London) = \frac{3}{4}$.

Let $sum^{M_i}(p)$ be the sum of the values of a measure attribute M_i over all the tuples matching p. Let $sum_r^{M_i}(p)$ be the sum of the values of a measure attribute M_i over all the tuples matching p and satisfying a predicate r over the measure attributes. For example, $sum^{Late}(*, *, London) = 20+15+19+7 = 61$ and $sum^{Late}_{Full=0}(*, *, London) = 7$.

We survey solutions to the following data exploration problem: given a dataset S, produce a set or a list of patterns P over the dimension attributes of S, as defined above, that summarize common properties of tuples sharing the same values of the measure attribute(s). The number of patterns in P should be limited to direct the user's attention to the most important or interesting regions of the data. This limit may be set explicitly by the user (in terms of the maximum number of patterns in P) or implicitly by retrieving the fewest possible patterns that jointly satisfy some property such as covering some fraction of the data.

This data exploration problem has the following applications.

- *Explaining the results of aggregate queries.* Suppose a data analyst issues the following query over Table 1: SELECT SUM(Late) FROM S. Suppose the analyst wishes to understand why the result, of 145, is so high. Here, interesting patterns are those which cover tuples that make a significant contribution to the result, i.e., those with a high $sum^{Late}()$ such as $(*, *, London)$. The analyst may then zoom into flights landing in London and investigate potential reasons for the lengthy delays.

- *Analyzing outliers and data quality issues.* Suppose we have a binary measure attribute denoting whether a given tuple contains an error or is an outlier. This attribute could be created manually by domain experts or automatically by identifying tuples that violate data quality rules or deviate from the expected distribution. We may wish to produce patterns that summarize the properties of erroneous tuples to help determine the root cause of data quality problems.

- *Feature selection and explainable AI.* Before building prediction models, a data scientist may explore interesting patterns to understand which dimension attributes are related to the measure attribute that is to be predicted. Furthermore, suppose a data analyst wants to understand how a black-box model makes classification decisions. Here, the dimension attributes are the features given to the model as input, and, as the measure attribute, the analyst records the predictions made by the model. The analyst may then want to find interesting patterns that explain the prediction decisions. For example, in Table 1, the pattern $(Mon, *, *)$ is associated with tuples having $Full = 0$, suggesting that flights scheduled on Mondays are usually not full[1].

3. Solutions

In this section, we provide a categorization of previous work on data exploration using multi-dimensional patterns based on the pattern properties and ranking strategies used for pattern selection. We categorize these properties into three types: those focusing on *coverage*, *contrast* and *information*. Table 2 categorizes the surveyed methods and lists their motivating applications, as mentioned in the corresponding papers.

Additionally, Table 3 lists the inputs and outputs of the surveyed methods. As explained in Section 2, these methods operate over datasets with multiple dimension attributes and a measure attribute that needs to be covered, contrasted or explained. MRI and Smart Drilldown also require a function that assigns pattern weights. CAPE is unique in its inputs in that it requires a specific pattern that can be thought of as a starting point for further exploration. In terms of outputs, most methods produce k best patterns according to some properties. Methods based on contrast identify k patterns with the highest contrast scores (details in Section 3.2). Methods based on coverage and information typically use greedy heuristics to solve an

[1]Model explanations may be global (to summarize how classification decisions are made) or local (to explain why a specific example was classified in a particular way); see [13] for a survey. The methods discussed in this paper are examples of global explanations.

Table 2: Methods surveyed

Method	Approach	Applications
CAPE [18]	Contrast	Explaining queries
Data Auditor [9]	Coverage	Data quality analysis
Data X-ray [24]	Contrast	Data quality analysis
DIFF [2]	Contrast	Outlier analysis
Explanation tables [7]	Information	Feature selection
Macrobase [1]	Contrast	Outlier analysis
MRI [5]	Coverage	Explaining queries over product ratings
RSExplain [20]	Contrast	Explaining queries
Scorpion [25]	Contrast	Outlier analysis
Shrink [11]	Information	Explaining queries
Smart Drilldown [15]	Coverage	Explaining queries, data cube exploration
SURPRISE [21]	Information	Explaining queries

underlying NP-hard problem related to maximizing the coverage or information content of the selected patterns. Thus, their outputs are *approximately* optimal with respect to the associated coverage, weighted coverage, or information metric.

3.1. Methods Based on Coverage

The goal of these methods is to identify patterns that *cover* tuples of interest; this may refer to covering tuples in the entire dataset, covering tuples with a given value of a measure attribute, or covering tuples that contribute to the result of a query. We discuss three coverage-based methods: Data Auditor [9], MRI [5], and Smart Drilldown [15].

3.1.1. Method Details

Suppose we want to cover tuples having $Full = 1$ in Table 1 to summarize the characteristics of full flights. A simple coverage-oriented approach is to sort the patterns according to $sup_{Full=1}()$ and output the top-ranking patterns. Ignoring $(*, *, *)$, which always covers everything but is not useful in data exploration, the top candidates are $(*, *, London)$ and $(*, *, Frankfurt)$, which cover three full flights each, followed by the following patterns that cover two such tuples each: $(Fri, *, *)$, $(Sun, *, *)$, $(Sat, *, *)$ and $(*, SF, *)$.

Table 3: Inputs and outputs of methods surveyed

Method	Input	Output
CAPE	Dataset, a pattern p	k patterns with the highest counterbalance score w.r.t. p
Data Auditor	Dataset, coverage thresholds $\theta_r(p)$	Fewest patterns that satisfy the coverage thresholds
Data X-ray	Dataset	k patterns with the highest diagnosis cost
DIFF	Dataset, contrast metric, support threshold	k patterns (that satisfy the support threshold) with the highest contrast
Explanation tables	Dataset	k most informative patterns
Macrobase	Dataset	k patterns with the highest risk ratio
MRI	Dataset, coverage threshold, pattern weighting function	k patterns that cover the required fraction of tuples with a minimal sum of weights
RSExplain	Dataset	k patterns with the highest intervention score
Scorpion	Dataset	k pattern with the highest influence score
Shrink	Dataset	k patterns that best summarize the distribution of the measure attribute
Smart Drilldown	Dataset, pattern weighting function	k patterns that maximize the product of coverage and sum of weights
SURPRISE	Dataset	k most informative patterns

There are two problems with this simple approach: there may be many patterns with a nonzero $sup_{Full=1}(p)$, and some of these patterns may also cover tuples with other values of the measure attribute (here, $Full = 0$). To reduce the size of the output and to ensure that each pattern co-occurs with the specified value of the measure attribute, Data Auditor solves the following set cover problem. Continuing with our example, Data Auditor requires a minimum threshold for $\theta_{Full=1}(p)$, i.e., the fraction of tuples covered by p that correspond to full flights. Suppose we require $\theta_r(p) \geq 0.75$. This threshold defines the candidate sets for the set cover problem, i.e., all the patterns p with $\theta_r(p) \geq 0.75$. The set cover objective is to select the fewest such patterns that together cover a specified fraction of tuples in S having $Full = 1$. Suppose this coverage fraction, which would again be set by the user, is 0.5. The goal is then to find the fewest patterns, as defined above, to cover half the full flights. Alternatively, Data Auditor may produce k patterns that (approximately) maximize coverage.

The set cover problem is NP-hard, and Data Auditor uses the standard greedy heuristic that achieves a logarithmic approximation ratio in the size of the solution: it iteratively chooses the pattern that covers the most uncovered tuples (having the desired value of the measure attribute), until the desired fraction of such tuples has been covered (or until k patterns have been chosen). In our example, the first pattern added to P is either $(*, *, London)$ or $(*, *, Frankfurt)$ – they each cover three full flights and their $\theta_{Full=1}(p)$ values are 0.75 each. Suppose the set cover algorithm selects $(*, *, Frankfurt)$. Since there are seven full flights in the dataset and our coverage threshold is 0.5, we need to cover one more full flight. In the next iteration, the pattern that (has $\theta_{Full=1}(p) \geq 0.75$ and) covers the most remaining full flights is $(*, *, London)$ and the algorithm terminates, with P consisting of these two patterns.

Data Auditor solves a set cover problem in which patterns are prioritized by their coverage (of tuples that have a given value of the measure attribute and have not yet been covered). The other two coverage-based methods take into account pattern weights in addition to coverage, as described below.

The next method, Smart Drilldown, produces k patterns. In each of the k iterations of the algorithm, the chosen pattern maximizes the following objective: the number of tuples not yet covered multiplied by the *weight* of the pattern corresponding to some measure of interestingness. One simple weighting function proposed in [15] is the number of non-star values in the pattern; i.e., more specific patterns are considered to be more desirable. In Table 1 for example, this weighting function prefers $(Tue, Chicago, LA)$ over $(Tue, *, *)$ – both of these patterns have a support of one, but the former has more non-star values.

The third method based on coverage, MRI, finds k patterns that cover a user-specified fraction of the data and satisfy additional properties related to the variance of the measure attribute within each pattern. Here, the weight of a pattern corresponds to this notion of variance, and patterns with smaller weights are preferred. The motivating example for MRI was to explain queries over product reviews, with the dimension attributes corresponding to information about the reviewers (such as their gender and age) and the numeric measure attribute corresponding to the average rating. Minimizing variance amounts to returning patterns (having high coverage and) describing reviewers with similar opinions. For example, when applying MRI to Table 1 with *Late* as the measure attribute, the pattern $(*, *, Frankfurt)$ is preferred over $(*, *, London)$. Both patterns cover four tuples, but the former has a lower variance of the *Late* attribute within the covered tuples.

Both Smart Drilldown and MRI prioritize patterns with high coverage (of tuples that have not yet been covered) and take pattern weights into account. However, while Smart Drilldown uses a modified version of the greedy set cover heuristic also used by Data Auditor, MRI uses a *hill climbing* heuristic. First, MRI selects k patterns at random. Next, it makes small changes to the patterns, if necessary, to ensure that the user-specified coverage threshold is satisfied. A small change to increase coverage may correspond to replacing one attribute value in a pattern with a '*'. Finally, MRI makes a second round of small changes to the patterns in an attempt to reduce the sum of their weights.

3.1.2. Pros and Cons

One advantage of coverage-based methods is conciseness: by design, they identify (approximately) the fewest patterns that cover the desired fraction of tuples of interest, or they produce k patterns that (approximately) maximize coverage. On the other hand, in Data Auditor, some trial-and-error may be required on the user's part to select good values for the two required thresholds. For example, a high value of θ will ensure that each selected pattern co-occurs mainly with the specified value of the measure attribute, but will disqualify more patterns from consideration, possibly leading to lower coverage. Similarly, MRI and Smart Drilldown require users to set parameters for coverage and other pattern properties.

3.2. Methods based on Contrast

This group of methods includes CAPE [18], Data X-ray [24], DIFF [2], Macrobase [1], RSExplain [20] and Scorpion [25]. Contrast-based methods often assume a binary measure attribute, and select patterns co-occurring with one value of the measure attribute but not the other. These patterns reflect the contrast between tuples having different values of the measure attribute.

One could argue that interpretable classifiers such as decision trees and rule-based methods (see, e.g., [16]) can also be used for contrast-based data exploration. These methods identify patterns of values of the feature attributes that have high discriminative power in terms of the class variable (in our case, the binary measure attribute). These patterns are therefore likely to provide contrast as well. However, classification algorithms usually focus on out-of-sample predictive power and include optimizations such as rule pruning to avoid overfitting. On the other hand, the methods covered in this survey focus explicitly on identifying a concise set of interesting fragments of the data for user exploration.

Contrast-based methods can also explain the results of aggregate queries. Consider the following query over Table 1: SELECT SUM(Late) FROM S WHERE Full=1. Here, one measure attribute corresponds to the quantity being aggregated. We then set the other (binary) measure attribute to one for all tuples that participate in the query (i.e., tuples that match the WHERE predicate), and we select patterns of tuples that contribute to the result of the query (but would not contribute had the query been issued against the other tuples in the dataset).

3.2.1. Method Details

DIFF is a recent solution that generalizes earlier contrast-based methods, including Data X-ray, Macrobase, RSExplain and Scorpion. The authors of DIFF observe that these methods all use a similar algorithmic framework but different contrast metrics. DIFF supports these different contrast metrics. To summarize the pattern mining framework used in DIFF to generalize prior work, a contrast metric is calculated for each candidate pattern and the highest-ranking patterns are returned. DIFF additionally implements several performance optimizations that will be discussed in Section 4. Below, we give several examples of contrast metrics originally used in earlier methods and now supported by DIFF. We again use the running example in Table 1, with *Full* as the binary measure attribute and *Late* as the additional numeric measure attribute when needed.

Risk ratio was originally used by Macrobase; a related metric called Diagnosis Cost is used by Data X-ray. It is the ratio of the following two probabilities: 1) the probability that a tuple with a particular value is covered by the given pattern, and 2) the probability that a tuple with this particular value occurs outside this pattern. In our example,

$$risk_{Full=1}(p) = \frac{\theta_{Full=1}(p)}{\frac{sup_{Full=1}(*,*,*) - sup_{Full=1}(p)}{(sup_{Full=1}(*,*,*) - (sup_{Full=1}(p)) + (sup_{Full=0}(*,*,*) - sup_{Full=0}(p))}}.$$

For instance, $risk_{Full=1}(*, *, London) = \frac{0.75}{0.4} = 1.875$, and $risk_{Full=1}(*, SF, *) = \frac{0.5}{0.5} = 1$. This indicates that $(*, *, London)$ represents full flights better than $(*, SF, *)$.

Mean shift computes the ratio of the mean of the measure attribute values co-occurring with the two values of the binary measure attribute. In our example,

$$mean^{Late}_{Full=1}(p) = \frac{sum^{Late}_{Full=1}(p)/sup_{Full=1}(p)}{sum^{Late}_{Full=0}(p)/sup_{Full=0}(p)}.$$

For instance, $mean^{Late}_{Full=1}(*, *, London) = \frac{54/3}{7/1} = 2.57$, indicating that full fights to London have delays that are 2.57 times longer than non-full flights to London.

Intervention was originally used by RSExplain; a related metric called Influence is used by Scorpion. It measures the ratio of contribution towards the numeric measure attribute for tuples occurring with the different values of the binary measure attribute. In our example,

$$intervention^{Late}_{Full=1}(p) = \frac{sum^{Late}_{Full=1}(*, *, *) - sum^{Late}_{Full=1}(p)}{sum^{Late}_{Full=0}(*, *, *) - sum^{Late}_{Full=0}(p)}.$$

For instance, $intervention^{Late}_{Full=1}(*, *, London) = \frac{108-54}{37-7} = 1.8$. In other words, if flights to London were removed from the dataset then full flights would have delays on average 1.8 times longer than non-full flights. On the other hand, $intervention_{Full=1}(*, SF, *) = \frac{108-35}{37-12} = 2.92$, meaning that removing flights departing from SF from the dataset would create a greater *contrast* between the delays of full and non-full flights.

Finally, we discuss CAPE. Given a specific pattern p as input, CAPE finds patterns whose tuples have measure attribute values that *counterbalance* those of p. Thus, instead of ranking patterns according to some contrast metric, CAPE ranks patterns according to a counterbalance metric with respect to p, and outputs the highest-ranking such patterns.

For example, suppose a user runs the following query on Table 1: SELECT AVG(Late) FROM T WHERE Day = 'Mon'. This query outputs the value 5.2 and touches tuples identified by the pattern $(Mon, *, *)$. This average delay is lower than the average delay in the entire table, which is 10.4. The user may then input this pattern to CAPE and request patterns that counterbalance the lower delays seen in this pattern. CAPE may then return patterns such as $(Fri, *, *)$ and $(Sat, *, *)$, whose average delays are higher than average (18 and 16, respectively). Thus, counterbalancing refers to finding related patterns whose measure attribute values are "outliers" in the other direction (in our example, higher than average) compared to the input pattern (in our example, lower than average).

In CAPE, the pattern mining algorithm has two main modules.

The first module finds regression relationships in the data; then counterbalancing can be used to find outliers with respect to these relationships. In our example above, counterbalancing was based on a trivial regression relationship involving the Late attribute, namely that AVG(Late)=10.5. Since the input pattern had lower than average delays, CAPE searched for patterns having higher than average delays. To see an intuitive example of a more complex regression pattern, paraphrased from [18], consider a database with authors and their publications such as DBLP. In this example, a regression pattern may hold for most authors such that they publish more papers over time. Here, an unusual (with respect to the identified regression relationship) pattern corresponding, say, to a lower-than-expected number of publications in a given year for given author, can be counterbalanced by finding patterns corresponding to years in which this author published more than the expected number of papers.

After identifying regression relationships, the second module finds patterns to counterbalance a given input pattern. Such patterns must satisfy two objectives. First, they must be "outliers" in the opposite direction to the input pattern with respect to some aggregate function over the measure attribute. Second, they must be "close" to the input pattern in terms of the values of the dimension attributes. In the DBLP example above, if the input pattern refers to, say, year 2015, then patterns with similar years, say 2014 or 2016, would be preferred for counterbalancing.

3.2.2. Pros and Cons

By design, contrast-based methods are useful when exploring differences between data subsets – something that coverage-based methods do not directly optimize for. On the other hand, contrast-based methods may not guarantee concise results. One exception is Data X-ray, which performs a

Table 4: An explanation table of size four for the binary measure attribute Full

Day	Origin	Dest.	Full
*	*	*	0.5
Mon	*	*	0
	*	London	0.75
	*	Frankfurt	0.75

set-cover-like operation on the extracted patterns as a post-processing step to eliminate redundant patterns.

3.3. Methods based on Information

Finally, we discuss three methods that select patterns based on the information they provide about the distribution of the measure attribute: Explanation Tables [7], SURPRISE [21] and Shrink [11]. We show two examples of explanation tables, one with a binary measure attribute and one with a numeric measure attribute, followed by a greedy heuristic to construct (almost) optimal explanation tables, and an overview of SURPRISE and Shrink.

3.3.1. Method Details

Table 4 shows an explanation table of size four (i.e., containing four patterns) for the binary measure attribute Full based on Table 1. In addition to values of the dimension attributes, each explanation table pattern also includes the fraction of matching tuples that have $Full = 1$. The first pattern in an explanation table is always the all-stars pattern, and, in this example, it indicates that half the flights in the entire dataset are full. The next pattern suggests that no flights on Mondays are full, and the last two patterns indicate that three-quarters of flights to London and Frankfurt are full.

In Table 5, we show an explanation table of size four for the measure attribute Late based on Table 1. Here, each pattern includes the average value of Late across its matching tuples. Again, we start with the all-stars pattern, which states that flights are 10.4 minutes late on average. The next pattern indicates that flights to London are 15.3 minutes late on average, and so on.

The greedy heuristic for constructing explanation tables used in [6, 7, 8] iteratively selects patterns that contain the most information about the distribution of the measure attribute. To do so, the algorithm maintains a

Table 5: An explanation table of size four for the numeric measure attribute Late

Day	Origin	Dest.	Late
*	*	*	10.4
	*	London	15.3
Fri	*	*	18
Sat	*	*	16

maximum-entropy estimate of the distribution based only on the patterns that have been added to the explanation table so far, and without assuming any other information; recall that the entropy of a random variable X with outcomes x_1 through x_n is defined as $E(X) = \sum_1^n -P(x_i) \log P(x_i)$. The algorithm also keeps track of the distance between the estimated distribution and the true distribution by computing their *Kullback-Leibler* (KL) divergence. Given two random variables, X and Y, with the same space of outcomes, x_1 through x_n and y_1 through y_n, respectively, the KL divergence of their distributions is defined as $KL(X, Y) = \sum_1^n P(x_i) \log \frac{P(x_i)}{P(y_i)}$. To quantify the information contained in a candidate pattern p, the algorithm computes the reduction in KL-divergence if p were to be added to the explanation table.

Returning to Table 4, the greedy algorithm starts by inserting the pattern $(*, *, *||0.5)$. At this point, knowing only this one piece of information, the maximum-entropy estimate of the distribution of $Full$ is to assign $Full = 0.5$ to every tuple in Table 1[2]. That is, knowing only that the average value of Full in the entire table is 0.5, we obtain maximum entropy by assigning 0.5 to each tuple. Next, it turns out that $(Mon, *, *||0)$ gives the greatest reduction in KL-divergence. Based on this new pattern, the maximum-entropy estimate for tuples 10 through 14 in Table 1 changes to $Full = 0$. This revision causes the estimates of the first nine tuples to change (from 0.5 to $\frac{7}{9}$) in order to maintain consistency with the first pattern, which asserts that $Full = 0.5$ on average over the entire table. Given the updated maximum-entropy estimate, the next pattern with the greatest reduction in KL-divergence is

[2] $Full$ is a binary attribute that can only be zero or one. However, for the purpose of measuring the divergence between the true and the estimated distributions, the maximum-entropy estimates are allowed to be real numbers between zero and one.

$(*, *, London||0.75)$, and so on.

Similar reasoning can explain how Table 5 was created. The first pattern asserts that flights are late by 10.4 minutes on average. Given this estimate, $(*, *, London||15.3)$ provides the most information about the distribution of *Late*. The maximum-entropy estimate of *Late* is now updated accordingly, That is, tuples 1, 4, 6 and 11, corresponding to flights to London, receive an estimate of 15.3, and the remaining tuples receive an estimate of 8.4 to maintain consistency with the first pattern. The next most-informative pattern is then selected, and so on.

SURPRISE is a similar method, whose goal is to identify surprising fragments of a dataset where the distribution of the measure attribute is different than what the user has seen so far. Suppose the user queries Table 1 and finds out that flights are 10.4 minutes late on average. SURPRISE finds the most informative *non-overlapping* and *contained* patterns, i.e., those which lead to the greatest reduction in KL-divergence between the true distribution of Late and the maximum-entropy estimated distribution. Restricting the output to such patterns makes it easier to update the estimated distribution. In our example, the most informative pattern is $(*, *, London)$ and its most informative subset is $(*, SF, London)$.

Finally, we discuss Shrink. The motivation behind Shrink was to create a new OLAP operator to reduce the size of a data cube after a user has drilled down to a fine-grained level. This is done by merging similar slices while balancing the tradeoff between the number of merged slices returned and accuracy of the corresponding aggregate function over the measure attribute. In the context of pattern-based exploration, we can position Shrink as a method that starts with a full data cube (i.e., all possible patterns) and produces k non-overlapping patterns that summarize the distribution of the measure attribute with (approximately) a minimal sum of squared errors.

Notably, Shrink explicitly allows patterns with disjunctions of values and dimension hierarchies. Table 6 shows an example of a summary with two patterns that may be considered by Shrink based on Table 1. We assume that the aggregate function is AVG(Late). Given the average values of the Late measure attribute reported by the summary, the sum of squared errors is 77.1.

Merging slices is done greedily. In each iteration, pairs of slices are chosen for merging if merging the corresponding aggregate values of the measure attributes leads to the smallest increase in sum of squared errors.

Table 6: A summary of size two considered by Shrink [11]

Day	Origin	Dest.	Late
Fri,Sun,Sat,Thu	*	*	15.4
Tue,Wed,Mon	*	*	5.3

3.3.2. Pros and Cons

By design, information-based methods produce *informative* patterns that highlight fragments of the data with surprising distributions of the measure attribute. However, these methods tend to be expensive, especially as the number of dimension attributes grows.

4. Performance Optimizations

A critical challenge in pattern-based data exploration arises from the size of the search space: the number of possible patterns is exponential in the number of dimension attributes. Coverage-based methods therefore deal with a large number of candidate patterns when constructing a set cover, contrast-based methods must compute contrast scores for many candidate patterns, and information-based methods must keep track of the information content of many patterns. In this section, we categorize frequently used performance optimizations to address these challenges and enable interactive data exploration.

We identify the following categories of optimizations: *top-down pruning*, *row pruning*, *column pruning* and *parallel processing*. Table 7 lists the methods that originally used these optimizations, indicated by a '*'; for brevity, we do not explicitly list methods whose contrast metrics are supported by DIFF since the optimizations implemented in DIFF also apply to those methods. As we will explain throughout this section, some optimizations may potentially be applicable to other methods, indicated by 'a' in Table 7. In particular, optimizations that will be experimentally evaluated in Section 5 are indicated by an 'E'.

4.1. Top-Down Pruning

Top-down pruning refers to traversing the space of candidate patterns from general (all-stars) to specific and pruning candidates along the way.

Table 7: Performance optimizations, methods that use them (indicated by a '*'), methods that may potentially use them (indicated by an 'a'), and new method-optimization combinations that will be tested experimentally in Section 5 (indicated by an 'E'). Methods are ordered by category, divided by horizontal lines: coverage-based, followed by contrast-based, followed by information-based.

Method	Top-down pruning	Row pruning	Column pruning	Parallel processing
Data Auditor [9]	*	a	E	a
Smart Drilldown [15]	*	*	a	a
CAPE [18]	a	a	*	a
DIFF [2]	*	E	*	*
Explanation tables [7]	E	*	*	*
Shrink [11]		a	a	a
SURPRISE [21]	a	a	a	a

The idea is similar to that of the Apriori algorithm for frequent itemset mining [3].

In the context of data exploration using patterns, we define the *ancestors* of a pattern p as all patterns p' such that $p \subseteq p'$, and the *descendants* of p as all patterns p' such that $p' \subseteq p$. We can generate the ancestors of p by replacing non-star values with stars. Similarly, we can generate the descendants of p by replacing one or more stars with values from the corresponding column. For example, in Table 1, the ancestors of $(*, SF, London)$ are $(*, *, London)$, $(*, SF, *)$ and $(*, *, *)$. Similarly, the descendants of $(*, *, London)$ are $(*, SF, London)$, $(Fri, *, London)$, $(Mon, SF, London)$, and so on.

Note that if p is a descendant of p' (equivalently, if p' is an ancestor of p) then $sup(p) \leq sup(p')$. DIFF exploits this observation by requiring the user to set a minimum support threshold for a candidate pattern. Then, similar to Apriori frequent itemset mining, DIFF traverses the space of candidate patterns from the top down. When a pattern p is encountered with a support below the threshold, p and all of its descendants can be ignored (i.e., their contrast measure will not be calculated). Since DIFF supports various contrast metrics used in other contrast-based methods, this performance optimization also applies to other contrast-based methods.

Coverage-based methods such as Data Auditor and Smart Drilldown also use top-down pruning. In each iteration of the greedy set cover heuristic, these methods select the next best pattern based on the number of uncovered tuples it covers (multiplied by the weight in case of Smart Drilldown). Thus, a performance bottleneck results from having to keep track of the number of uncovered tuples that can be covered by the candidate patterns – this changes in every iteration, whenever a new pattern is added to the solution.

Here, an important observation is that a descendant of a pattern p cannot cover more uncovered elements than p. Data Auditor exploits this observation and does not generate all the patterns that serve as input to greedy set cover beforehand. Instead, patterns are generated on-demand, only after all of their ancestors have already been considered. For example, a pattern such as $(Fri, *, London)$ would only be generated after all of its ancestors - including $(Fri, *, *)$ and $(*, *, London)$ - have already been considered. Until then, $(Fri, *, London)$ can be safely ignored: it cannot cover more elements than its ancestors and therefore is guaranteed to not be selected by the greedy set cover heuristic at this time. This optimization can greatly reduce the number of generated patterns whose coverage (of uncovered elements) must be re-computed while constructing the solution.

Top-down pruning has not been applied to information-based methods. One problem is that a descendant pattern may be more informative, even if its support is lower. In Section 5, we will investigate whether the approach used in DIFF, which is to require the user to set a minimum support threshold for a candidate pattern, is beneficial for information-based methods.

Finally, as we indicated in Table 7, there does not appear to be an obvious way to apply top-down pruning to Shrink. During pattern mining, Shrink starts with all possible patterns and iteratively merges patterns until the final result contains only k merged patterns, for some user-supplied value of k. This is a bottom-up approach. Furthermore, ignoring patterns with low support does not appear to be helpful since these patterns would not be merged and instead would appear individually in the final output. This would increase the size of the output, which is the opposite of the intended goal of reducing the number of patterns.

4.2. Row Pruning

A simple example of row pruning is sampling: we draw a random sample from the input dataset and run a data exploration algorithm on the sample instead of the entire dataset. In principle, sampling applies to all the methods discussed in this survey. In particular, sampling is used by information-based methods such as Explanation Tables to reduce the number of patterns whose information gain must be computed, and by coverage-based methods such as Smart Drilldown to reduce the number of patterns whose coverage must be computed. Furthermore, sampling may speed up the computation of information gain or coverage of candidate patterns since there is less data to scan. For example, the explanation table construction algorithm in [6, 7, 8] draws a random sample in every iteration. Next, the set of candidate patterns corresponds to only those patterns that have a non-zero support in the sample. The intuition is that patterns with frequently occurring combinations of dimension attribute values are likely to be sampled and also likely to contain information about the distribution of the measure attribute.

However, a disadvantage of generating patterns from a sample is that any pattern statistics calculated from a sample, such as contrast scores or information gain, may not be accurate with respect to the full dataset. This is in contrast to top-down pruning, which does not affect the quality of the output: candidates are ignored only if they are guaranteed to not be included in the output because their support is below the minimum support threshold set by the user. Flashlight, which is one of the explanation table algorithms proposed in [6], suggests the following compromise: sampling is used only to restrict the candidate pattern space (to those with non-zero support in the sample), but the information gain of candidate patterns is computed over the full dataset, not the sample. This way, the resulting explanation table is more informative since the patterns reflect the distribution of the measure attribute in the entire dataset rather than the distribution within the sample.

4.3. Column Pruning

This category of optimizations removes some columns from consideration in order to reduce the space of candidate patterns. We identified two examples of column pruning: removing correlated columns and limiting pattern size. Again, both of these optimizations apply to all the methods surveyed in this paper.

In terms of removing correlated columns, both CAPE and DIFF include a pre-processing step that removes attributes that are functionally determined by other attributes. The modified dataset is then used for exploration.

Limiting pattern size refers to limiting the number of non-star values that appear in a pattern, which again reduces the space of candidate patterns. This optimization is used by CAPE, DIFF and SIRUM [8], which is one of the explanation table construction algorithms.

4.4. Parallel Processing

Parallel processing is used by DIFF and explanation tables (SIRUM). In a contrast-based method such as DIFF, there are many candidate patterns whose contrast measure needs to be computed. These computations can be done independently for each pattern, and therefore can be parallelized. In explanation table construction, every iteration requires the computation of information gain of candidate patterns. This can also be parallelized, but only within an iteration, not across iterations, since information gain of the remaining patterns may change when a new pattern is added to the explanation table. Finally, in coverage-based methods, it may be possible to parallelize the computation of coverage of candidate patterns.

5. Experiments

In Section 4, we suggested that some performance optimizations originally proposed in the context of one method may apply to other methods. In this section, we experimentally verify this claim, by applying optimizations originally proposed for a method in one category to a representative method from another category. In particular, we evaluate the following *new* combinations of methods and optimizations.

1. DIFF (contrast-based) uses top-down pruning by ignoring patterns whose support is below a user-specified threshold. To evaluate the impact of this optimization on information-based methods, we add this optimization to the Flashlight algorithm for explanation table construction. In other words, in each iteration, the modified Flashlight algorithm does not compute the information gain of any patterns whose support is below the threshold.
2. Row pruning (sampling) was originally used in Flashlight (information-based) and Smart Drilldown (coverage-based). To evaluate the impact of this optimization on contrast-based methods, we compare the performance on DIFF on a full dataset and DIFF on a random sample of a dataset.
3. DIFF (contrast-based) and SIRUM (information-based) limit the number of non-star values that a pattern may use, which is an example of column pruning. We apply this optimization to the Data Auditor coverage-based method. To do this, we modify the Data Auditor algorithm to ignore patterns with more than a user-specified number of non-star values.

Note that we do not investigate new applications of parallel processing optimizations. These kinds of optimization usually require a redesign of the underlying algorithm (e.g., the SIRUM algorithm for parallel construction of explanation tables [8]), which is outside the scope of this survey and is an interesting direction for future work.

5.1. Experimental Setup

Overview: Experiments were performed in Ubuntu Linux 16.04, on a device with Intel Core i7-6700HQ 2.60 GHz processor and 12 GB of RAM. Each experiment (i.e., each combination of method, dataset, and parameter settings) was repeated three times. For each experiment, we report the average running time across the three runs, as well as a measure of the goodness or quality of the output, such as coverage percentage or information gain.

Data: We use the following datasets:

- *Flight Upgrade*: This dataset includes 5794 United Airlines ticket records, with 7 categorical flight information attributes (origin, destination, airplane model, etc.), and a binary measure attribute determining whether or not the passenger had their ticket upgraded to business class. The dataset was used to evaluate explanation tables in previous work [7] and was obtained from the authors of the corresponding paper. We use this dataset in Experiments 1 and 3 to evaluate optimizations of the Flashlight algorithm for explanation tables and Data Auditor, respectively.

- *Adult Income*: There are 32561 records in this US Census dataset, where each record includes 8 categorical attributes about the adult person demographics (age, sex, education, etc.), and a binary measure attribute denoting whether the income level of the person is above or below $50,000. This dataset is accessible from the UCI repository[3], and we use this dataset for Experiment 2 to test optimizations of DIFF. Note that we do not use the smaller Flight Upgrade dataset to test DIFF because DIFF queries run much faster than the other methods, and thus measuring DIFF runtimes on small datasets does not show any meaningful differences.

[3]https://archive.ics.uci.edu/ml/datasets/adult

- *Bank Marketing*: This dataset, which can be accessed from UCI repository[4], includes records of bank marketing campaign outcomes, including 10 categorical attributes about the demographic information of each customer and the campaign contacts with the customer. The binary measure attribute determines whether the campaign was effective and the customer subscribed to the service that was advertised. We use the smaller version of the dataset with 4119 records for Experiments 1 and 3 (explanation tables and Data Auditor), and the larger full version with 41188 records for Experiment 2 (DIFF).

Source code: We obtained the DIFF source code from the project Github page[5]. We obtained the Flashlight source code, written in C++, from the authors of [23], and the Data Auditor code, also written in C++, from the authors of [9].

5.2. Experiment 1: Impact of top-down pruning on information-based methods

In this experiment, we compare the original Flashlight algorithm with a modified algorithm that ignores patterns whose support is below a user-specified threshold. This reduces the space of candidate patterns, at the expense of solution quality since some of the pruned patterns may have been informative. We thus investigate the tradeoff between running time and the information gain of the resulting explanation tables for various minimum support thresholds: 0.005, 0.01, 0.05. We fix the size of the explanation table at ten patterns, and we set the sample size for pattern generation to 16 (which was shown to be effective in previous work [6]).

Figure 1 displays the results of this experiment on the Flight (left) and Bank (right) datasets. The plots show how the information gain (y-axes) and execution time (x-axes) of the Flashlight algorithm change for different minimum support thresholds (the blue points), compared to the original algorithm without any top-down pruning (the black point).

Our results show that top-down pruning does not reduce the running time significantly, while the information gain of the resulting explanation tables degrades. Further analysis showed that the main performance bottleneck in the Flashlight algorithm is the generation of candidate patterns, and the

[4]http://archive.ics.uci.edu/ml/datasets/Bank+Marketing
[5]https://github.com/stanford-futuredata/macrobase

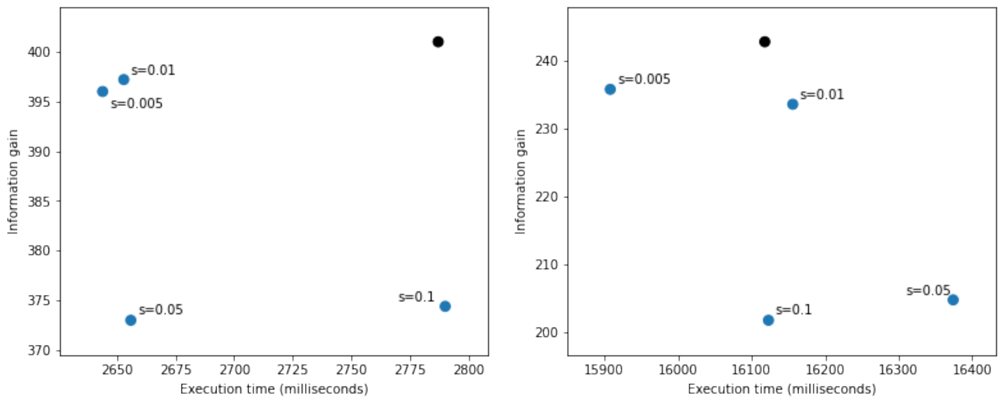

Figure 1: Experiment 1 results using Flashlight on Flight (left) and Bank (right) datasets

performance gain from not having to compute the information gain of some of the generated patterns (i.e., those below the given support threshold) is minimal. Furthermore, we noticed that patterns with large support increase the time taken by the algorithm to update the maximum-entropy estimate of the distribution of the measure attribute via iterative scaling. Thus, the time savings resulting from not having to compute the information gain of some patterns are often offset by the extra time taken to update the estimated distribution.

5.3. Experiment 2: Impact of row pruning on contrast-based methods

Next, we run DIFF on an entire dataset and on samples of the dataset. Reducing the sample size should make DIFF faster, at the expense of a larger error in the computed contrast scores. We thus evaluate the tradeoff between running time and the accuracy of contrast scores computed from a sample.

We test the following sample fractions of the full dataset: 0.01, 0.05, 0.1 and 0.2. From the contrast metrics supported by DIFF, we test the risk ratio and the mean shift in two separate sets of experiments (recall Section 3.2.1). For the risk ratio scores, we compute the contrast between subsets of the data with different values of the binary measure (denoting high income in the Adult Income dataset or marketing campaign success in the Bank Marketing dataset). For the mean shift scores, we use numeric measure attributes (denoting the number of working hours per week in the Adult Income dataset and the number of times the customer was contacted in the Bank Marketing dataset). We use the default DIFF parameters for

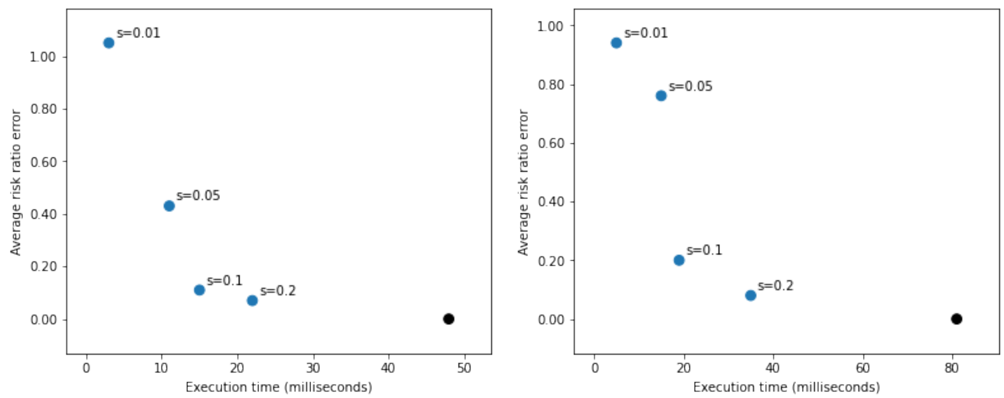

Figure 2: Experiment 2 results using DIFF risk ratio score on Income (left) and larger Bank (right) datasets

minimum support (10%) and minimum contrast score (1.5) thresholds.

To compare the DIFF contrast scores for full and sampled datasets, for each sample size, we compute the difference between the contrast scores of every pattern common between the full dataset result and the sampled result, and we take the average of these differences as the overall average error.

The results of the experiments with DIFF risk ratio score for the Income (left) and larger Bank (right) datasets are shown in Figure 2. The mean shift results are shown in Figure 3. The average contrast score errors for different sample size fractions are shown (blue points), compared with the full dataset (the black point).

Based on the four plots, we conclude that sampling is effective in reducing the running time (x-axes), with the average contrast score errors becoming quite small at 0.1 and 0.2 sample fractions (y-axes). However, smaller samples appear to increase the error without a significant corresponding improvement in running time.

5.4. Experiment 3: Impact of column pruning based on pattern size on coverage-based methods

Finally, we compare the original Data Auditor algorithm with a modified version that limits the number of non-star values a pattern may have. This optimization should reduce the space of candidate patterns and therefore the running time, possibly at the expense of solution quality (more patterns may be required to reach the desired coverage). However, patterns with

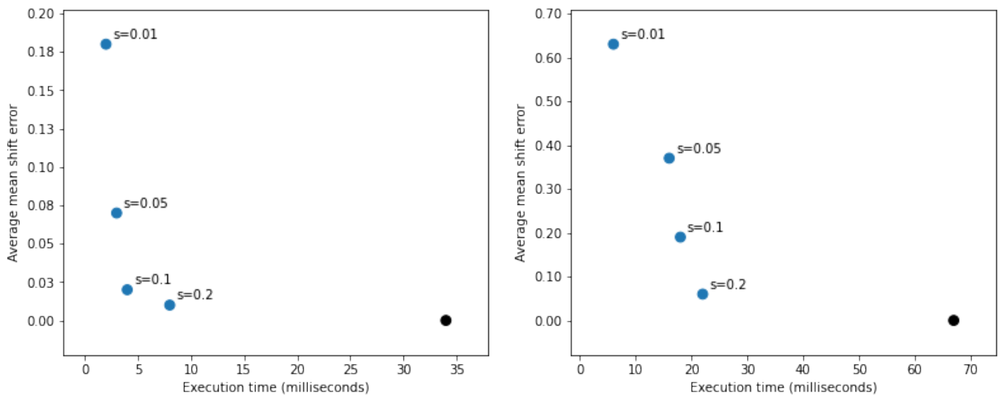

Figure 3: Experiment 2 results using DIFF mean shift score on Income (left) and larger Bank (right) datasets

many non-star values are expected to have low coverage, meaning that this optimization should be quite effective.

We test the following thresholds for pattern size, i.e., the maximum number of non-star values in a pattern: 1, 2, 3 and 4. We set the $\theta_r(p)$ threshold to 0.9. By default, Data Auditor terminates when the generated patterns collectively have the user-specified coverage. However, in order to compare the results obtained using different maximum non-star thresholds based on their total coverage, we modified the code to output a fixed maximum number of patterns, which in our experiments is set to 10, similar to the first experiment with explanation tables.

Figure 4 shows the results for the Flight (left) and Bank (right) datasets. The blue points represent the coverage and running time for different values of the pattern size threshold (denoted by m), while the black point shows the result of the original Data Auditor algorithm without any pattern pruning.

Limiting the number of non-star attributes significantly reduces the execution time (by multiple orders of magnitude; x-axes) without degrading the total coverage significantly (y-axes). In the Flight dataset, setting the threshold to two results in the best trade-off between performance and accuracy, while in the Bank dataset, a higher threshold of four provides significant improvements in running time with a small decrease in coverage. In both plots, we see that having only one non-star attribute in the patterns reduces the total coverage significantly, likely because there are few such patterns that meet the $\theta_r(p)$ threshold, and therefore there are few candidate sets for

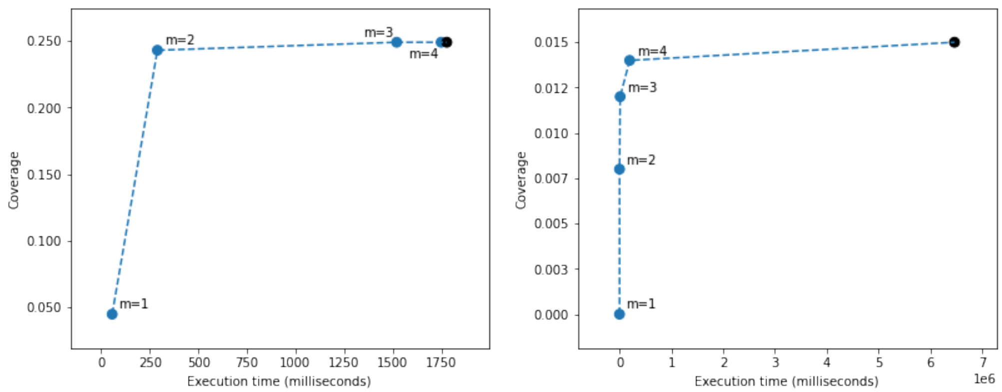

Figure 4: Experiment 3 results using Data Auditor on Flight (left) and Bank (right) datasets

the set cover solution produced by Data Auditor. In general, finding a good value of m for a given dataset is an interesting open problem.

6. Conclusions and Open Problems

We surveyed recent data exploration methods that extract interesting or informative fragments of the data, represented as patterns over the dimension attributes. We categorized these methods according to the properties of patterns they select, and we identified and experimentally evaluated frequently used performance optimizations. Our experimental study should be of interest to researchers working in the area of pattern-based data exploration as it suggests that many existing performance optimizations may apply to newly developed techniques. Specifically, our experiments showed that using a sample of a dataset is effective in improving the performance of contrast-based methods supported by DIFF, and we observed a performance gain after applying a limit on pattern size in the Data Auditor coverage-based method.

Below, we offer suggestions for future work in this area.

Benchmarks: A performance comparison of contrast-based methods implemented within the DIFF frameworks appears in [2]. In terms of effectiveness, prior work reports that methods based on information provide more information about the distribution of the measure attribute than coverage-based methods [6, 7]; similarly, methods based on contrast provide more precise outlier explanations than methods based on coverage [24]. Some approaches were also evaluated through user studies against simple baselines [18]. An interesting direction for future work is to develop a benchmark to highlight the effectiveness of various types of methods in various applications.

New applications: Popular motivating applications that guided the development of prior work were outlier and data error analysis, as well as query result explanation. Recent interest in explainable AI motivates further studies on exploring the behaviour of black-box machine learning models such as neural networks using multi-dimensional patterns, as was suggested in [7]. Since deep learning methods have been successful in the context of unstructured data such as text, images and graphs, future research should investigate new ways of formulating interpretable patterns over these high-dimensional unstructured datasets.

Correlated measure attributes: Another characteristic of prior work is that it usually formulates exploration problems involving a single measure attribute. Pattern-based exploration of multiple measure attributes is an interesting area for future work.

Feature reduction: In terms of performance and scalability, the large number of possible patterns remains a challenge for many methods, especially those based on information which cannot leverage Apriori-like pruning strategies. This is an important challenge for interactive methods that allow users to continuously issue new exploration tasks. As a result, some techniques such as DIFF limit the number of dimension attributes for use in patterns and discard redundant dimension attributes such as those functionally determined by other attributes. Distributed versions of some methods, including DIFF [2] and Explanation Tables [8], have also been proposed to parallelize the search for interesting patterns. In machine learning, there exists a variety of dimension reduction methods such as Principal Component Analysis (PCA) and word embeddings. However, these methods are not known for being interpretable and thus their suitability for pattern-based exploration requires further study.

Bringing order to dimension attributes: Much of the previous work considers categorical dimension attributes. However, there exist methods for covering a multi-dimensional dataset using *hyper-rectangles* corresponding to intervals over numeric dimension attributes [17], there exists a method to cover data anomalies using intervals over numeric features [27], and explanation tables have recently been extended to support ordinal and numeric dimension attributes [23]. These extensions further increase the space of candidate patterns and require additional performance optimizations. For example, returning to Table 1, the Day attribute may lead to additional patterns with ranges or intervals such as $([Mon{-}Fri], *, *)$ or $([Sat{-}Sun], *, *)$. Techniques used to construct optimal histograms and optimal decision trees may help to optimize the discovery of these types of patterns.

Exploring data evolution: Finally, recent work motivates the need for tools to explore how data (and metadata) change over time [4]. Here, patterns may summarize fragments of the data that have changed recently or are updated often.

References

[1] F. Abuzaid, P. Bailis, J. Ding, E. Gan, S. Madden, D. Narayanan, K. Rong, S. Suri: MacroBase: Prioritizing Attention in Fast Data. ACM Trans. Database Syst. 43(4): 15:1-15:45 (2018)

[2] F. Abuzaid, P. Kraft, S. Suri, E. Gan, E. Xu, A. Shenoy, A. Anathanaraya, J. Sheu, E. Meijer, X. Wu, J. F. Naughton, P. Bailis, M. Zaharia: DIFF: A Relational Interface for Large-Scale Data Explanation. PVLDB 12(4): 419-432 (2018)

[3] R. Agrawal, R. Srikant: Fast Algorithms for Mining Association Rules in Large Databases. VLDB 1994: 487-499

[4] T. Bleifuss, L. Bornemann, T. Johnson, D. V. Kalashnikov, F. Naumann, D. Srivastava: Exploring Change - A New Dimension of Data Analytics. PVLDB 12(2): 85-98 (2018)

[5] M. Das, S. Amer-Yahia, G. Das, C. Yu: MRI: Meaningful Interpretations of Collaborative Ratings. PVLDB 4(11): 1063-1074 (2011)

[6] K. El Gebaly, P. Agrawal, L. Golab, F. Korn, D. Srivastava: Interpretable and Informative Explanations of Outcomes. PVLDB 8(1): 61-72 (2014)

[7] K. El Gebaly, G. Feng, L. Golab, F. Korn, D. Srivastava: Explanation Tables. IEEE Data Eng. Bull. 41(3): 43-51 (2018)

[8] G. Feng, L. Golab, D. Srivastava: Scalable Informative Rule Mining. ICDE 2017: 437-448

[9] L. Golab, H. J. Karloff, F. Korn, D. Srivastava: Data Auditor: Exploring Data Quality and Semantics using Pattern Tableaux. PVLDB 3(2): 1641-1644 (2010)

[10] L. Golab, D. Srivastava: Exploring Data using Patterns: A Survey and Open Problems. DOLAP 2021: 116-120

[11] M. Golfarelli, S. Graziani, S. Rizzi: Shrink: An OLAP operation for balancing precision and size of pivot tables. Data Knowl. Eng. 93: 19-41 (2014)

[12] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh: Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals. Data Min. Knowl. Discov. 1(1): 29-53 (1997)

[13] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi: A Survey of Methods for Explaining Black Box Models. ACM Comput. Surv. 51(5): 93:1-93:42 (2019)

[14] S. Idreos, O. Papaemmanouil, S. Chaudhuri: Overview of Data Exploration Techniques. SIGMOD Conference 2015: 277-281

[15] M. Joglekar, H. Garcia-Molina, A. G. Parameswaran: Interactive data exploration with smart drill-down. ICDE 2016: 906-917

[16] H. Lakkaraju, S. H. Bach, J. Leskovec: Interpretable Decision Sets: A Joint Framework for Description and Prediction. KDD 2016: 1675-1684

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

[18] Z. Miao, Q. Zeng, C. Li, B. Glavic, O. Kennedy, S. Roy: CAPE: Explaining Outliers by Counterbalancing. PVLDB 12(12): 1806-1809 (2019)

[19] X. Qin, Y. Luo, N. Tang, G. Li: Making data visualization more efficient and effective: a survey. VLDB J. 29(1): 93-117 (2020)

[20] S. Roy, D. Suciu: A formal approach to finding explanations for database queries. SIGMOD Conference 2014: 1579-1590

[21] S. Sarawagi: User-cognizant multidimensional analysis. VLDB J. 10(2-3): 224-239 (2001)

[22] P. Vassiliadis, P. Marcel: The Road to Highlights is Paved with Good Intentions: Envisioning a Paradigm Shift in OLAP Modeling. DOLAP 2018.

[23] M. Vollmer, L. Golab, K. Bohm, D. Srivastava: Informative Summarization of Numeric Data. SSDBM 2019: 97-108

[24] X. Wang, X. L. Dong, A. Meliou: Data X-Ray: A Diagnostic Tool for Data Errors. SIGMOD Conference 2015: 1231-1245

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

[26] J. Xu Yu, L. Qin, L. Chang: Keyword Search in Databases. Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2009

[27] H. Zhang, Y. Diao, A. Meliou: EXstream: Explaining Anomalies in Event Stream Monitoring. EDBT 2017: 156-167