INDIANA: an Interactive System for Assisting Database Exploration

Antonio Giuzio[a], Giansalvatore Mecca[a], Elisa Quintarelli[b], Manuel Roveri[b], Donatello Santoro[a], Letizia Tanca[b]

[a] *Università della Basilicata – C.da Macchia Romana, 85100 Potenza – Italy*
[b] *Politecnico di Milano – Piazza Leonardo Da Vinci 32, 20133 Milano, Italy*

Abstract

We propose INDIANA, a system conceived to support a novel paradigm of database exploration. INDIANA assists the users who are interested in gaining insights about a database though an interactive and incremental process, like a conversation that does not happen in natural language. During this process, the system iteratively provides the user with some features of the data that might be "interesting" from the statistical viewpoint, receiving some feedbacks that are later used by the system to refine the features provided to the user in the next step. A key ability of INDIANA is to assist "data enthusiastic" users (i.e., inexperienced or casual users) in the exploration of transactional databases in an interactive way. For this purpose, we develop a number of novel, statistically-grounded algorithms to support the interactive exploration of the database. We report an in-depth experimental evaluation to show that the proposed system guarantees a very good trade-off between accuracy and scalability, and a user study that supports the claim that the system is effective in real-world database-exploration tasks.

Keywords: database exploration, features, statistics, sampling

1. Introduction

The state-of-the-art opportunity to generate, collect and store digital data, at unprecedented volumes and speed, is still compromised by the limited human ability to understand and extract information from them. As a consequence, the classical ways for users to access data, i.e. through a search engine or by querying a database, have today become insufficient when compared to the new needs of data exploration and interpretation typical of an *information-producing* society. In fact, a request to a search engine typically produces a huge, indigestible collection of documents, and also by querying a DBMS the user receives all the (possibly enormous) mass of data that satisfies the specified criteria. With so large result datasets, the user can only start browsing data trying to make sense of them, e.g., by relying on a ranking function or sorting them according to a chosen notion of pertinency. This mechanism is satisfactory when the intention of the search or query is to locate a specific information, such as the address of a restaurant or the salaries of all the employees with a certain position, but ranked retrieval and querying are no longer adequate when the need of the user is to (quickly) explore a large collection of data, extracting and presenting interesting information in a condensed yet illuminating way.

Exploring data may mean to investigate and seek inspiration, compare data, use them to take decisions, verify a research hypothesis, or just browse documents and learn something new: all these relevant and challenging activities cannot be carried out through the traditional search and query mechanisms.

Most of the tools that are nowadays available for similar purposes are typically tailored to professional users, i.e., *data scientists* having a deep knowledge of the domain of interest and generally mastering data science disciplines such as mathematics, statistics, computer science, computer engineering, and computer programming. Instead, our aim is

Email addresses: antonio.giuzio@gmail.com (Antonio Giuzio), giansalvatore.mecca@unibas.it (Giansalvatore Mecca), elisa.quintarelli@polimi.it (Elisa Quintarelli), manuel.roveri@polimi.it (Manuel Roveri), donatello.santoro@gmail.com (Donatello Santoro), letizia.tanca@polimi.it (Letizia Tanca)

to support also inexperienced or casual *data enthusiastic* users [17, 25], like journalists, investors, or politicians. In our mind, non-technical users can gain great advantages from exploring the data to extract new and relevant knowledge instead of reading records by the ton. To achieve this goal, these users need the help of a sophisticated system that, starting from scratch or from simple inputs, guides and assists them along an *exploration path*. Similarly to a (more expert) friend, such a system would stimulate the user to explore the database by iteratively providing summarized representations of the interesting data and continuously highlighting the relevant properties of current and past query answers. This would help the user in getting an idea of the data in order to go to a new exploratory step, or formulate a query to locate a specific information[20].

With this idea in mind, in this paper we propose a paradigm for database exploration motivated by the vision of *exploratory computing* [7]. The term *data exploration* [19] is rather general, and its meaning might differ in different scientific areas. For instance, in the field of statistics, Tukey [29] introduced *exploratory data analysis*, referring to the activity where users explore manually a data collection in many ways, possibly with the support of graphical tools like box plots or histograms, and gain knowledge also from the way data are displayed. However, this kind of exploratory data analysis still assumes that the user understands at least the basics of statistics. Differently, in our approach the process brings to mind a human-to-human dialogue[7], in which a person experiences a process of investigation, inspiration-seeking, comparison-making and suggestion, by means of a tool for getting the gist of *big collections of semantically rich data*, which typically hide precious knowledge behind their complexity.

A fundamental aspect, typically considered when designing a database exploration tool, is the *relevance of an answer.* Capturing the right notion of relevance for a specific user means interpreting her/his expectations. More specifically, the result of an exploration step might be relevant because it meets the user's expectations or, on the contrary, because it differs from them, hence surprising the user. Having to deal with the interpretation of the user expectations, relevance remains largely a subjective factor, hence an immediately-related research challenge is to predict users' interests and anticipations in order to issue the most relevant (possibly serendipitous) answer. Here we do not delve into the related fields of personalization and recommendations, but adopt a (orthogonal) notion of relevance that does not make use of previous information on the user, relying instead on (statistically) *significant aspects of the data themselves*. In addition to these challenges, we identified the following primary issues in designing exploration systems:

- *Scalability and Response Time are clearly fundamental requirements*: we want the system to provide prompt answers to the user, and the exploration to be perceived as "smooth", without long pauses to identify relevant features. Computing times should scale nicely with large databases. In fact, we investigate several techniques to speed-up the exploration of the database, e.g., *sampling* mechanisms.

- *Accurate statistics are critical quality factor for data enthusiasts*: we want the system to correctly find the features that can be interesting for the user, as well as to provide accurate results about the attribute values' statistics. This means that high accuracy is required and only a small degree of approximation can be tolerated when presenting results to the user.

- *False positive detections*: a recent study [6] has raised the concern that many of the recently proposed tools for exploring and visualizing databases (see the related work section for a more detailed discussion) suffer from the risk of "false discovery", i.e., selecting features only due to the so-called error of Type I in statistical tests (false positive detections). The paper highlights the fact that, considering large numbers of such visualizations, some are proposed as interesting regardless of whether or not the underlying phenomenon is statistically relevant.

To address all the aforementioned issues, this paper proposes the INDIANA system, supporting database exploration by means of an interactive process[1] that guides non-expert users in the navigation of a large database, based on expressive queries and a sophisticated, statistics-based notion of relevance, without the use of any precomputed data structure. The distinguishing features and contributions of the work are:

[1] A virtual machine running INDIANA is available at `https://www.dropbox.com/s/o6wipy07bmgfru5/UbuntuIndiana.zip?dl=0`.

1. We propose the exploration model at the basis of our approach, based on paths within *a lattice of conjunctive views*. These represent the possible connections among the various tables and the selection of potentially interesting information that can be suggested to the users. Our model is general enough to be applied to most data models and query languages that have been proposed for data management in the last few years.
2. We develop a statistical notion of *relevance*, and identify *relevant features*, i.e., attributes in the view lattice that are interesting according to the exploration process of the user. Intuitively, our notion of relevance is based on the idea of analysing and comparing the statistical characteristics of features, e.g., their empirical distributions or sample moments to find the ones that exhibit peculiarities. To this end, we consider several statistical hypothesis tests.
3. We introduce several different strategies to implement this interaction and show how our hybrid implementation strategy represents the best trade-off between accuracy of results and scalability.
4. We implement the INDIANA system in a working prototype. The system features a graphical user interface that can be easily customized to the dataset at hand, and supports seamless interactions with users.
5. Using the developed prototype, we conducted an in-depth experimental evaluation, with both accuracy and scalability tests on various synthetic datasets, and a user study on two exploration tasks that confirms the effectiveness of our approach.

The proposed system brings a significant enhancements with respect to previous proposals, in the sense that, as described in the next sections, our feature-selection algorithm is rooted in solid statistical hypothesis tests. The adoption of such tests strongly reduces the risk of proposing features that are not statistically relevant. We report an experimental comparison that confirms this comment.

The paper is organized as follows. We present a motivating example in the next section and discuss related work in Section 3. The exploration model is introduced in Section 4 while the suite of algorithms implemented within the system is described in Sections 5 and 6. Statistical tests are detailed in Section 7. The e xperimental results are in Section 8, and, finally, the conclusions are drawn in Section 9.

2. A Motivating Example

We illustrate the process of exploring a large and semantically rich dataset through the example of a database containing the recordings of the fictional fitness tracker AcmeBand, a wrist-worn smartband that continuously records user steps during the day and tracks user sleep during the night.

The INDIANA user is granted access to the measurements of a large fragment of AcmeBand users. In this example, for the sake of simplicity, we shall assume that the database is a relational one, and focuses on conjunctive queries as the query language of reference. However, we want to emphasize that the techniques proposed in this work can be applied to any data model that is based on the primitives of object collection, attribute and attribute value, and object reference. We emphasize that our approach may incorporate the majority of data models that have been proposed in the last few years to model rich data. In our simplified case, the database has the following structure:

(i) a AcmeUser(id, name, sex, age, cityId) table with user data;

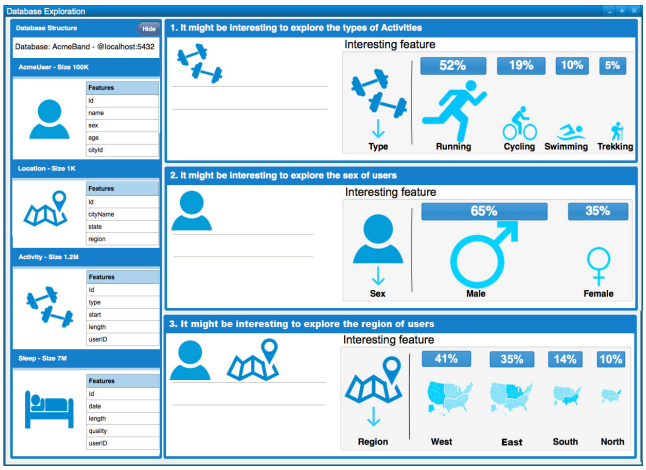
(ii) a Location(id, cityName, state, region) table to record location data about users (here region may be east, west, north or south);

(iii) an Activity(id, type, start, length, userId) table to record step counts for user activities of the various kinds (like walks, runs, cycling etc.);

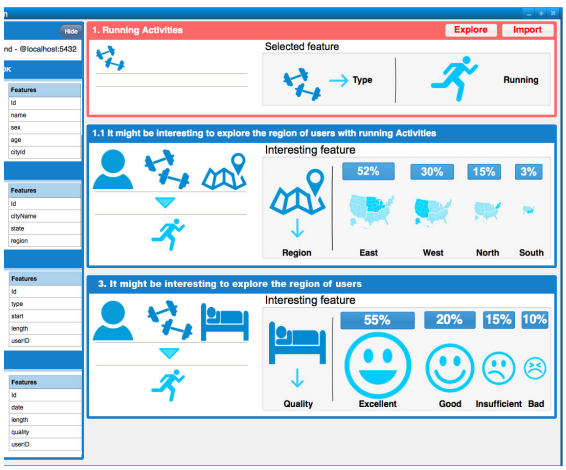
(iv) a Sleep(id, date, length, quality, userId) table to record user sleep and its quality (like deep sleep, restless etc.).

Note that the database may be quite large, even if the number of AcmeBand users is limited and the timeframe for activities and sleep restricted. Our casual exploratory user intends to acquire some knowledge about fitness levels and sleep habits of this fragment of AcmeBand users. We do not assume any a-priori knowledge about a database query language, nor the availability of pre-computed summaries as the ones that are typically used in Online Analytical Processing (OLAP) applications. On the contrary, the system we have in mind should be able to guide and support our users throughout their (casual) information-seeking tasks.

A sample interaction between INDIANA and a user is depicted in Figure 1. We envision this process as an *interaction* between the user and the system, where s/he provides some initial hints about her/his interests, and INDIANA



Step 0: Initial suggestions



Step 1: User selects "Running Activities", the system provides further suggestions

Figure 1: User interface: an example of database exploration through INDIANA

suggests "potentially interesting perspectives" over the data that may help to refine the search. Intuitively, these perspectives are captured by the notion of *feature*, that is an attribute in the database, either of a base table, or of a view over the base tables (this notion will be clarified in Section 4). During the interactive process, the system suggests to the user some *relevant features*. Given our sample database, to trigger the exploration, INDIANA suggests some initial relevant features, for example along the following paths (see Figure 1):

$(\mathcal{S}_1)$ "It might be interesting to explore the types of activities. In fact, in this dataset: *running* is the most frequent activity (over 50%), *trekking* the least frequent one (less than 5%)";

$(\mathcal{S}_2)$ "It might be interesting to explore the sex of users. In fact: more than 65% of users are *male*";

$(\mathcal{S}_3)$ "It might be interesting to explore the regions of user's locations. In fact: 41% of the users are from the *west* of the country, 35% *east*, 14% *south*, 10% *north*".

Assume the user selects the suggestion $\mathcal{S}_1$. This choice is processed by the system as the first step in the interactive process: the user has selected table Activity and relevant feature type and INDIANA shows a subset of values for the type attribute sorted by frequency. Then, the user may pick one or more of these values. Let us assume s/he selects *running* and this is interpreted by INDIANA as some interest in the *concept* of "running activities", internally modeled by the system *as a view over the database*:

$$\sigma_{\mathsf{type}=\mathit{running}}(\mathsf{Activity}).$$

Following the user selection, the new view is generated and added to the process. The addition of a new concept triggers a new interaction, with the system trying to suggest new and relevant hints. To do this, it may be necessary to compute further views: for instance, INDIANA finds that a relevant feature is related to the region of the runners (i.e., users that perform running activities) by computing this new view:

$$\sigma_{\mathsf{type}=\mathit{running}}(\mathsf{Location} \bowtie \mathsf{AcmeUser} \bowtie \mathsf{Activity})$$

Among the potentially relevant features the system will suggest that (see Figure 1):

$(\mathcal{S}_{1.1})$ "It might be interesting to explore the region of users with *running* activities. In fact: while users are primarily located in the west, 52% of users with *running* activities are in the *east*, and only 3% in the *south*". The user will select the suggestion $(\mathcal{S}_{1.1})$, and then pick up value *south*. The process is triggered again, and INDIANA suggests that:

$(\mathcal{S}_{1.1.1})$ "It might be interesting to explore the sex of users with *running* activities in the *south* region. In fact: while 35% of users are women, less than 5% of users with *running* activities in the *south* are *women*". It can be seen that

the discovery of this new relevant feature requires to compute the following view and add the corresponding concept:

$$\sigma_{\mathsf{type}=\mathit{running},\mathsf{region}=\mathit{south},\mathsf{sex}=\mathit{female}}(\mathsf{Location} \bowtie \mathsf{AcmeUser} \bowtie \mathsf{Activity})$$

After s/he has selected the suggestion ($\mathcal{S}_{1.1.1}$), the user is effectively exploring the set of tuples corresponding to female runners in the south. S/he may decide to ask the system for further advices, or browse the actual database records, or use an externally computed distribution of values to look for relevant information. Assume, for example, that s/he is interested in studying the quality of sleep. S/he downloads from a Web site an Excel sheet stating that over 60% of the users complain about the quality of their sleep. When these data are imported into the system, INDIANA suggests that, unexpectedly, 85% of sleep periods of southern women that run are of good quality. Having learned new insight on the data, the user is satisfied.

3. Related Work

The literature on database exploration, recently surveyed in [19], involves different topics and aspects, notably: assisting the users' navigation to find interesting objects, transforming the data to provide different ways to familiarize with them, improving the efficiency of the data exploration process. Differently from other approaches (e.g. [12]), where a natural language interface has been developed to help user in the exploration task, INDIANA relies on a visual and textual interaction between user and system.

Exploration and Visualization Interfaces Interactive queries, faceted search in Information Retrieval (IR) and Human-Computer Interaction (HCI), and approximate answers in relational databases are all exploration-oriented research areas that, albeit different in nature, share some points of contact with our proposal.

For example, in [22] the authors, motivated by similar concerns as ours, propose some directions to improve the user experience when querying scientific databases – a typical example of large masses of structured data whence the user would like to get an initial idea without necessarily reading the millions of tuples that are in a query result. In this work the authors concentrate on redesigning some features of the DBMS in such a way as to better support incremental or stepwise query processing, statistical analysis and data mining, to build data summaries or alternative query suggestion. While these ideas are all close to ours in suggesting various means for exploration, and thus might well be classified under the general umbrella of exploratory computing, our work concentrates more on the two basic concepts of *interaction* and *relevance*, both of which place major emphasis on the user experience.

Also in the way of interactive queries, AIDE [13] iteratively guides the users towards interesting data areas, and predicts a query that retrieves their objects of interest by using relevance feedback on database samples to model user interests. QueRIE [14] is a system that assists users in the interactive exploration of large databases. QueRIE monitors the user's querying behaviour with the aim of identifying users with similar needs; then, on the basis of users similarities, it recommends interesting queries to the current user. In interactive querying, the relevance of a result is computed mostly on the basis of the previous experience with the same user or similar ones, as in most recommender systems.

ForeCache [5] is a tool for suggesting browsing patterns on multi-dimensional numerical data to users; the recommendations are provided with different levels of granularities by aggregating data, and are strongly focused on the user's interests about previously explored data.

This is one of the main principles of relevance by personalization, and relies strongly on previous system logs; by contrast, we want to suggest serendipitous and relevant data, and our method to assess relevance, based on the differences with other query results, does not need previous knowledge about the users.

The other fundamental research line is *faceted search*, also called faceted navigation, and its variations, which (often visually) support the access to information items based on *facets* [30]. Facets are initially classified according to a given taxonomy. The user can browse the dataset making use of the facets and of their values (e.g. facet *Activity*, or the value *"running"*) and will inspect the dataset of items that satisfy the selection criteria. As an example, [10] proposes a dynamic faceted search system for discovering textual or structured data. The aim is to dynamically choose the most interesting attributes and provide the user with (precomputed) elementary aggregates thereof. This work is close to OLAP exploration [27], where the interestingness is defined as a kind of serendipity of the aggregated values w.r.t. a given expectation. This last research is strongly oriented to multi-dimensional, again precomputed, analytical queries, while we concentrate on on-line computation of the relevance of the features. In the OLAP direction much

work has also been placed in the way of building summaries of various kinds, like histograms, sketches, etc., or precomputed aggregations in the form of materialized views or data cubes.

All these works, both in the area of faceted search and of OLAP exploration, tend to provide ad-hoc results, built for a specific, pre-envisaged query (usually on a collection of documents); instead, our aim is to provide users with the possibility to form their own ideas about a complex database, by "plunging" into the data and coming out multiple times, through of a sequence of articulated conjunctive queries supported by the fast online computation of complex statistics.

The quest for an effective user interaction immediately raises the important issue of developing intuitive visualization techniques. An exploratory interface should support appealing, synthetic visualization of the query answers[25]. Several interfaces have been proposed with the purpose of helping users in the visualization and exploration process; for instance, [4] introduces an interactive visualization system for the reduction of query results, while [23] focuses on rapidly generating approximate visualizations that preserve crucial properties of interest to analysts, such as trends. Moreover, to automate data-driven exploration, visualization recommender systems, such as [31, 33, 34, 11], have been proposed: they suggest visualizations to provide insights of datasets and refine previous visualizations encountered in the exploration process. SeeDB [31] suggests the visualizations that are most relevant to provide insights about the user query; the notion of relevance is computed on the basis of the deviation from the user query. For a more in-depth comparison of our proposal to SeeDB we refer the reader to the experimental evaluation section. Voyager [33] provides automatically-generated visualizations by means of charts based on data variation (different variable selections and transformations). Voyager 2 [34] extends [33] by integrating both manual and automatic chart specification; the user can specify partial views (i.e., with wildcards) on the analyzed dataset and the system can automatically recommend charts to help users to begin the exploration process or to refine previous performed exploration steps. Foresight [11] is focused on providing visual insights from high-dimensional datasets and thus their main goal is also in this case complementary to ours.

Summarization and Approximation Handling big data imposes to work with data summaries, the size of which must be large enough to estimate statistical parameters and distributions, but manageable from the computation viewpoint. To solve this problem many summarization techniques can be explored. Some works, for instance [2], have proposed to extract knowledge by means of data mining techniques, and to use it to intensionally describe sets in an approximate way. To summarize the contents of large relations and permit efficient retrieval of approximate query results, the authors of [26] have used histograms, presenting a histogram algebra for efficiently executing queries on the histograms instead of on the data, and estimating the quality of the approximate answers. The latter research approaches can be used to support query response during the interactive exploration.

Another technique for approximate processing is adopted by DICE[21], a framework that enables interactive cube exploration by allowing the user to explore facets of the data cube, trading off accuracy for interactive response-times thanks to data sampling.

A further critical point related to computation is finding effective pruning techniques for the view lattice: not all node-paths are interesting from the user viewpoint, and the ones that fail to satisfy this requirement should be discarded as soon as possible in the exploration process. This again introduces interesting problems related to detecting the different relevance that a feature might have depending on the users: a feature may be relevant if it is *different from*, or maybe *close to*, the user's expectations; in turn, the users' expectations may derive from their previous background, common knowledge, previous exploration of other portions of the database, etc.

In order to provide efficient visual recommendations, Foresight[11] adopts the sketching paradigm, whose goal is provide approximate computations of values of interest without requiring the storage of all the acquired data. This paradigm, very appropriate the analysis of data streams, relies on hashing mechanisms and behaves similarly to sampling; however it is generally characterized by a high computational complexity and, due to this approximation, traditional statistical mechanisms (such as hypothesis tests) cannot be directly considered and applied.

Responsiveness Responsiveness is related to a research area that is traditional for databases. DBMSs build and maintain *histograms* representing the distribution of attribute values to estimate the (possibly joint) selectivity of attribute values for use in query optimization [16]. Fast and incremental histogram computation is a good example of a technique that can be effectively employed for speeding up the assessment of relevance in a conversation step.

Fast exploration times can also be achieved by caching data sets which are likely to be used by a user's future query. Data prefetching has been proposed for different types of queries; [28] discusses an indexing technique on past

users' explored data.

Efficiency in ForeCache [5] is achieved by means of pre-fetching techniques and of a main-memory cache that stores information relevant for the current user (on the base of the past browsing activities) to speedup server-side performance, where different levels of aggregations on data are collected.

Fast Statistical Operators and Queries The previous points bring us to another fundamental requirement: the availability of fast operators to compute statistical properties of the data. Surprisingly, despite years of data-mining research, there are very few research proposals towards the goal of fast-computing statistical parameters, and comparing them. Subgroup discovery [24, 18], endeavors to discover the maximal subgroups of a set of objects that are statistically "most interesting" with respect to a property of interest. Subgroup discovery has been proposed in many flavors, differing as for the type of encompassed target variable, the description language, the adopted quality measure and the search strategy. Again, we believe that some of its techniques might be profitably adopted to assess the relevance of features in each of our exploration steps.

BlinkDB [1] is a parallel approximate query processing framework for interactive database querying, providing real-time approximate answers with statistical error guarantees and proposing sampling for speeding purposes. There are a few points of contact with INDIANA – as an example, INDIANA also extensively uses sampling for the purpose of selecting features, as it will be discussed in Section 5. However, the two systems have quite different scopes and approaches. BlinkDB only considers aggregation queries, with no joins, and, more important, the samples are prepared off-line based on the envisaged workload. On the contrary, samples in INDIANA are extracted on-line for comparing distributions to each other.

Some preliminary ideas about the application of exploratory computing techniques to data exploration were discussed in a previous short paper [8], which focused mainly on the vision behind our approach, and essentially contained no technical details. Here, we develop the full technical machinery needed to implement that vision in a concrete system.

4. Modeling Explorations

Assume we are given a relational database schema $\mathcal{R} = \{R_1, \ldots, R_k\}$, and an instance I of $\mathcal{R}$, defined as usual. From the technical viewpoint, the interaction between system and user is modeled as a lattice $\mathcal{L}$, in which each node is a *view* over the database described intensionally as a conjunctive query Q, and extensionally by a *tuple set* $\mathcal{T}^Q$ defined as the result of Q over I, i.e., $\mathcal{T}^Q = Q(I)$. Part of the lattice of views for our running example is depicted in Figure 2. As it can be seen, nodes in the lattice correspond to conjunctive queries. In fact, our main purpose is to identify attributes of lattice nodes that have "interesting" sets of values. To do this, we rely on a statistical comparison of value distributions. One may wonder how aggregations are involved in this process. To start, aggregations, i.e., COUNT queries, stand at the core of our approach to construct value distributions and compare them to each other using statistical tests. This approach is rooted in statistical theory, and gives strength to our notion of "interestingness" of an attribute. Furthermore, the model is extensible, in the sense that, in addition to comparing value-distributions, attributes can also be compared to each other using other aggregation-based features, like their average, maximum or minimum value. This is a fairly straightforward extension of our model of relevance.

In what follows, with a slight abuse of notation, we blur the distinction between the query Q and the corresponding node in the lattice. Similarly, given a lattice $\mathcal{L}$ and a node $Q \in \mathcal{L}$, when clear from the context, we will use the word *feature* to refer to an attribute A in $\mathcal{T}^Q$. Again, to simplify the notation, we will often omit the reference to the conjunctive query Q when referring to the tuple set $\mathcal{T}^Q$.

Our goal in this section is twofold:

(i) we want to formalize the algorithm that is used by INDIANA in order to dynamically construct the lattice that models the interaction between user and system;

(ii) we want to formalize the notion of a *relevant feature* A, based on the statistical properties of the distribution of values of feature A in the current instance I.

We discuss these aspects in the following.

Lattice Construction First, to start the interaction, the system populates the lattice with a set of initial views. These will typically correspond to the database tables themselves, and joins thereof according to foreign keys. Broadly

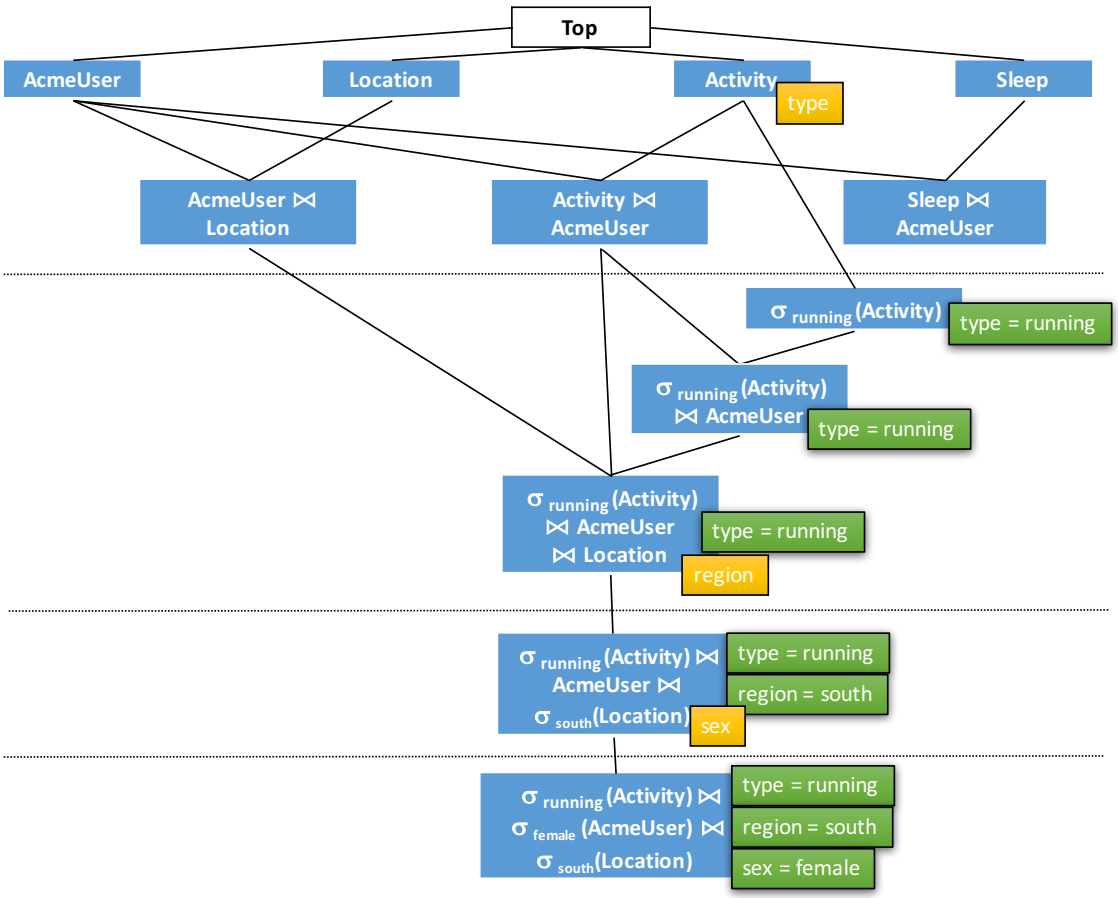


Figure 2: A portion of the final lattice

speaking, each view node can be seen also as a *concept* of a conversation, described by a sentence in natural language. For example, the node corresponding to the Activity table represents the "activity" concept (a *root* concept, wherefrom the system may proactively start the interaction), while the join of AcmeUser with Location represents the concept of "user location", and so on.

Formally speaking, we fix a parameter max_{joins}, corresponding to the maximum join cardinality at each step, and initialize the lattice $\mathcal{L}$ by adding:

(a) one *base* node Q_i for each base table $R_i \in \mathcal{R}$;

(b) one *join* node Q_j for each query obtained by joining at most max_{joins} base nodes $Q_1 \ldots Q_k$ according to *(key, foreign key)* pairs.

To simplify the presentation, in the following we assume that $max_{joins} = 1$. Suggestions by the system correspond to relevant features, i.e., interesting attributes within lattice nodes, as formalized in the following paragraphs. Whenever the user chooses to explore an interesting feature, s/he is selecting: (a) a concept, i.e., a node Q in the lattice, e.g., "users with running activities"; (b) one of its attributes A, e.g., "region"; (c) one or more values $v_1, \ldots, v_k$ for that attribute, e.g., "south".

By doing this, the system generates a new node in the lattice, corresponding to query $\sigma_{A \in \{v_1,\ldots,v_k\}}(Q)$. In our example, by choosing to explore the feature "the region of users with running activities" the following node is generated:

$$\pi_{region}(\sigma_{\text{type}=running}(\text{Location} \bowtie \text{AcmeUser} \bowtie \text{Activity}))$$

while, when progressing further by choosing the "southern region", the new user-generated node is query Q':

$$\sigma_{\text{type}=running,\text{region}=south}(\text{Location} \bowtie \text{AcmeUser} \bowtie \text{Activity})$$

To formalize the lattice structure, we reason about the relationship among conjunctive queries: we say that the tuple-set $\mathcal{T}^Q$ returned by a query Q over $\mathcal{R}$ *derives* from the tuple-set $\mathcal{T}^{Q'}$ returned by a query Q' if Q is obtained from Q' by adding one or more selections, one or more projections, or one or more joins. We denote this by $Q \preceq Q'$, and, accordingly, we say $\mathcal{T}^Q \preceq \mathcal{T}^{Q'}$. In our example, the view AcmeUser $\bowtie$ Location derives from the views AcmeUser and Location. Similarly, view $\sigma_{type=running}$(AcmeUser $\bowtie$ Activity) derives from AcmeUser and Activity, but also from view $\sigma_{type=running}$(Activity). If we add, as a convention, a *top view* Q_{top} such that $Q \preceq Q_{top}$ for any query Q, it is possible to see that the relationship "derives from" among views over a schema $\mathcal{R}$ induces a join-semilattice.

Notice that the lattice also encodes a taxonomy: whenever the user imposes a restriction over the current view (concept), this account for the identification of one of its "sub-concepts".

Identifying Relevant Features In order to identify relevant features, we focus on attributes of the lattice nodes and analyze the statistical distribution of their values, i.e., the empirical distribution. Given node n in $\mathcal{L}$, let $\mathcal{T}$ be the associated tuple-set and, for an attribute A, let us denote by $d_{A,\mathcal{T}}$ the empirical distribution of values in $\pi_A(\mathcal{T})$ for A and $\mathcal{T}$. An important remark is that, since we want to reason about the frequency of values of a feature, we will rely on bag semantics for projections.

More specifically, we adopt a comparative notion of relevance based on a statistical nature. For each node Q and attribute A, we identify a number of distributions that are "uninteresting", called *reference* distributions (empirically or theoretically-modelled) defined as $unint_{\pi_A(Q)} = \{d_1, \ldots, d_k\}$ (see below for details). We deem A as a *relevant feature* when $d_{A,\mathcal{T}}$ is different from at least one of reference distributions $d_1, \ldots, d_k$. To achieve this goal, we assume the availability of a statistical similarity test, denoted by STATTEST(d, d'), to compare empirical distributions d, d', as discussed in Section 7.

If A is a categorical attribute, i.e., it has a small number of possible values, computing the empirical distribution is straightforward. If, on the contrary, A has a non-finite domain – like, for example, integer attributes – then we group values in buckets of a fixed size.

Now we need to formalize the reference distributions $unint_{\pi_A(Q)}$ for a feature A of Q. This is done separately for base nodes and nodes at lower levels.

If Q is a base node, we assume as reference distributions one or more known distributions, e.g., the uniform distribution, or the Normal distribution (the system offers a library to choose from). Assume the system has been configured in order to use the uniform distribution for base nodes. In our example, the type attribute of Activity is considered as potentially relevant (suggestion $\mathcal{S}_1$ above), since its distribution shows a significant statistical difference, according to STATTEST(), w.r.t. the uniform one.

For nodes that are lower in the lattice the definition of reference distribution is slightly more sophisticated. In fact, whenever a query Q derives from Q', we want to keep track of the relationships between each attribute of $Q(I)$ and the corresponding one in $Q'(I)$. In order to uniquely identify attributes within views, we denote attribute A in table R by the name R.A[2]. We say that attribute R.A in Q *matches* attribute R.A of any Q' that is an ancestor or descendant of Q in the query lattice.

As mentioned in [6], both base and lower nodes might introduce false relevant detections induced by the confidence parameter of the hypothesis tests. To mitigate this problem, once relevant features have been identified, we rank them according to a dissimilarity measure to list those that exhibit the largest difference. In our specific case, we rank all the identified features according to the Hellinger Distance; other dissimilarity measures could have been considered as well. Afterwards, only the subset of features characterized by the largest dissimilarity are selected. Alternatively, we could have considered a confidence correction for multiple hypothesis tests (e.g., the Bonferroni correction) but, as pointed out in [6], this might be too conservative when the number of hypothesis tests running in parallel is very high.

Now we are able to define the set of reference distributions for A of Q as the set of distributions of its matching attributes: we say that attribute R.A of node $Q(I)$ is *relevant* if its distribution $d_{A,\mathcal{T}}$ is statistically different from the distribution of the matching attribute from any ancestor node. An example is given in Suggestion $\mathcal{S}_{1.1}$.

[2]For the sake of simplicity here we do not consider self-joins, i.e., joins of a table with itself. In this way each attribute A of table R may appear only once within a view Q. Note that, with a bit more work, the definition can be extended to handle self-joins.

5. The Feature Discovery Algorithm: the INDIANA system

This paper introduces INDIANA, a system to assist database explorations following the model, described in Section 4. Since an important motivation for this work is to support casual users and "data enthusiasts", a basic assumption of our study is that users should be allowed to conduct explorations by working with a transactional database. As a consequence, we assume that the lattice is constructed dynamically, based on user interactions, on a database that may be concurrently updated, without relying on the possibility to construct a data warehouse or pre-compute views.

We emphasize that pre-computing views is not a viable option in this context, even if a data warehouse is available. Consider for example the simple lattice in Figure 2. We see that there are three main categories of nodes: (i) nodes corresponding to base tables – i.e., the level 0 of the lattice; (ii) nodes corresponding to joins thereof – the level 1; (iii) nodes at higher levels, corresponding to selections over nodes in the previous two levels, generated through the conversation with the user. In fact, in order to completely materialize the lattice, in addition to joins, one would need to materialize a selection node for each possible combination of values of any relevant feature, i.e., a number of nodes that is exponential in the size of the database instance[3]. This is not a viable option in online interactive systems as the one we are here suggesting.

Another important consequence we can draw from the discussion above is that, from the scalability viewpoint, running the *base iteration* – i.e., finding relevant features in nodes at levels 0 and 1 – would yield to a significant improvement of the process: in comparison to lower levels, this is the only moment in which the entire database needs to be analyzed, hence execution time is critical to provide quick responses to users.

We now introduce the main algorithms that stand at the core of the INDIANA system. We primarily focus on the aspects of query execution and computation of empirical distributions for identifying relevant features.

Let us consider the base iteration of the exploration process, consisting in the generation of the nodes of levels 0 and 1 of the lattice, and let us analyze their attributes in order to identify the first set of relevant features to be shown to the user.

The pseudocode of the base iteration is shown in Algorithm 1. We summarize its main steps as follows (see Figure 2 for one example):

(i) *Building level–0 and level–1 nodes*: the algorithm starts with an empty lattice $\mathcal{L}$, and an empty set of relevant feature $relevantFeatures$. It first generates lattice nodes for the base tables, and the joins of (key, foreign key) pairs (lines 3–10). This allows to define *levels 0 and 1* of the lattice.

(ii) *Computing empirical distributions*: For each node n_i in the lattice, the system computes the corresponding tuple-set by running the associated conjunctive query. For a base table this is a simple table scan, while it is a join for the rest of the nodes (line 12). Subsequently, tuple-sets are analyzed to identify relevant features: for each attribute A_i of a tuple-set $\mathcal{T}_i$, the empirical distribution of values, d_{A_i} is computed by means of COMPUTEDISTRIBUTION() (line 17). Procedure TOSKIP is used to discard attributes that should not be analyzed: for example, it usually discards keys and foreign keys that are supposed to be purely syntactic references and carry no semantics. Notice that these are known in advance, so that this step does not require any statistical analysis.

(iii) *Running statistical tests*: Distribution d_{A_i} is compared to the set of relevant distributions for attribute A_i (lines 18–24) as described in Section 7. To achieve this goal we rely on statistical test STATTEST() able to evaluate whether d_{A_i} is statistically different from (at least one of) the relevant distributions of A_i and, in this case (line 20), attribute A_i of tuple-set $\mathcal{T}_i$ is added to the relevant features, $relevantFeatures$. Once $relevantFeatures$ has been defined, features stored therein are ranked according to the Hellinger Distance as described above.

(iv) *Presenting features to the user*: The systems presents the set of relevant features, i.e., $relevantFeatures$, along with a bunch of promising values for each of them and a textual description of the lattice node, based on the associated conjunctive query (lines 27–29).

Once the user has selected attribute A_i of node n_j as a feature, and values $v_1, \ldots, v_l$ as the values to explore, a new node is generated, corresponding to $\sigma_{A_i \in \{v_1,\ldots,v_l\}}(n_j)$. This is then joined with tables in the upper levels according to foreign keys that were not joined so far. This generates *level 2* of the lattice, and the process is iterated according to steps (ii)–(iv) above.

Implementation strategies for these operations are discussed in the next section.

[3]To be more precise, the number of nodes generated by a relevant feature A_i of tuple-set $\mathcal{T}_j$ is exponential in the size of the active domain of each A_i in $\mathcal{T}_j$.

Algorithm 1 ALGORITHM BASEITERATION($\mathcal{R}, I$)

$\mathcal{L} = \emptyset$
$relevantFeatures = \emptyset$
for all $R_i \in \mathcal{R}$ **do**
  $n_i = R_i$
  $\mathcal{L} = \mathcal{L} \cup n_i$
  for all foreign key $R_i.A$ referencing $R_j.B$ **do**
    $n_{ij} = R_i \bowtie_{A=B} R_j$
    $\mathcal{L} = \mathcal{L} \cup \{n_{ij}\}$
  end for
end for
for all $n_i \in \mathcal{L}$ **do**
  $\mathcal{T}_{n_i} =$ RUNQUERY(n_i, I)
  for all attributes A_i of n_i **do**
    if TOSKIP(A_i) **then**
      continue
    end if
    $d_{A_i} =$ COMPUTEDISTRIBUTION$(A_i, \mathcal{T}_{n_i})$
    $relevantDists_{A_i} =$ FINDRELEVANTDISTS$(A_i, \mathcal{T}_{n_i})$
    for all distributions $d_j \in relevantDists_{A_i}$ **do**
      if STATTEST(d_i, d_j) **then**
        $relevantFeatures = relevantFeatures \cup \{A_i\}$
        break
      end if
    end for
  end for
end for
RANK *relevantFeatures* according to Hellinger Distance
for all feature $A_i \in relevantFeatures$ **do**
  SHOWFEATURETOUSER(A_i)
end for
return $\mathcal{L}$

6. Computing Distributions

Computing probability distributions for attributes in the lattice in a fast and accurate way is a primary requirement of our system. As usual, we blur the distinction between a lattice node, n, and the corresponding conjunctive query. We assume that query n has been executed to generate tuple-set $\mathcal{T}^n$. Computing the probability distribution for values of attribute A_i within $\mathcal{T}$ aims at building the histogram of value frequencies.

For non-categorical attributes, the processing goes along the same lines. We treat string attributes as if they were categorical. For numerical values, before computing probabilities, we need to group values in buckets of a fixed size, between the minimum and maximum value.

In the next subsections we introduce three different families of algorithms that can be used to this end.

6.1. Algorithms PUREDB *and* PUREDB-H

The algorithms of the first family are purely based on the use of the database query language.

6.1.1. Algorithm PUREDB

There is a straightforward algorithm to perform this step by using the DBMS. We call this algorithm PUREDB, as it entirely relies on the database engine to perform the computation. It has a very good accuracy. However, as we will discuss shortly, it hardly scales to large databases, and, as a consequence, it will be considered exclusively as a baseline we compare faster variants to. Let us first consider a categorical attribute, i.e., an attribute with a finite set of possible values. The PUREDB algorithm works as follows: it materializes tuple-set $\mathcal{T}$ as a table, and then runs the following query:

SELECT A_i, count(A_i) FROM $\mathcal{T}$ GROUP BY A_i

To build the actual probability distribution, the system loads the query result in main memory to learn all value frequencies. We denote by $numOcc(v_j)$ the number of occurrences of value v_j in $\mathcal{T}.A_i$. The system computes the total number of values, $size_{A_i}$ as $\Sigma_{v_j \in \mathcal{T}.A_i}(numOcc(v_j))$, and then the frequency of each value v_j as $numOcc(v_j)/size_{A_i}$.

It can be seen that this is optimal from the accuracy viewpoint: in fact, it returns the exact empirical probability distribution for A_i. The drawback is that it requires to compute a large number of queries.

6.1.2. Algorithm PUREDB-H

Before we move to the discussion of faster, alternative algorithms based on sampling mechanisms, let us briefly discuss a variant of the PUREDB algorithm. We emphasize that the DBMS natively stores some histograms within the database catalog, in the form of database statistics. Algorithm PUREDB-H queries the catalog to extract histograms that the DBMS has previously materialized for query optimization purposes. This approach is faster, but it has several shortcomings:

(i) These histograms are not always complete. On the contrary, the DBMS often stores frequencies for just few representative values. This obviously reduces the accuracy of the results.

(ii) Histograms are available for base tables only and, in this case, they might not be up-to-date. In fact, frequent concurrent transactions may alter the distribution of values in such a way that the DBMS cannot keep the catalog up-to-date with the data. To enforce syncing, an explicit command must be issued to analyze the fresh tables and update the statistics accordingly. This step is necessary for lattice nodes that correspond to non-base tables: these tuple-sets need to be materialized and analyzed in order to extract the statistics of interest.

We discuss the performances of PUREDB and PUREDB-H in Section 8.

6.2. Sampling Algorithms

The previous section highlighted the drawbacks of the PUREDB PUREDB-H algorithm. To overcome these drawbacks, we introduce a novel algorithm called SAMPLINGMM that can be summarized as follows:

(i) The query for each node n is run once, and a random sample, SAMPLE($\mathcal{T}$) of its tuple-set $\mathcal{T}$ is extracted and loaded into main memory.

(ii) For all attributes A_i of node n, the probability distribution is estimated in main memory by means of randomly extracted samples.

The obvious advantage over PUREDB is provided in performance. In fact, with this approach, there is no need to materialize the tuples as a temporary table and, in addition, the sample size is usually much smaller than the one of the whole tuple-set, i.e., typically 1% of the whole size. Finally, the histograms can be computed in main memory using fast hash-based data structures.

The significant reduction in the complexity/execution times for the estimation of the distribution and running of statistical tests comes at the expense of a possible drop in the possibility to correctly identify relevant features: in fact, accuracy is as good as the representativeness of the samples extracted from the tuple-set.

In the attempt to strike a balance between accuracy and scalability, we explore three different sampling strategies in the coming paragraphs.

In the following we assume we are considering the node n in the lattice – recall that by n we also refer to the associated conjunctive query – and we intend to extract a sample of size k_n defined as the percentage of the associated tuple-set. Size k_n corresponds usually with a percentage of the size of the entire tuple-set, K_n, e.g. 1%. A preliminary task is to estimate K_n, in order to derive k_n.

6.2.1. Determining Sample Sizes

In principle, discovering the result size for a conjunctive query over a database requires to run a separate size-discovery query with a $count(*)$ before actually sampling the result.

We notice, however, that in our approach this can be done with a very limited overhead, since we can get good estimates of the size of most tuple-sets associated with lattice nodes. In fact:

(i) Level-0 nodes correspond to base tables, whose initial size we assume to be known since we make the reasonable hypothesis that the system runs a size-discovery query for each database table at startup.

(ii) Level-1 nodes are joins of base tables on foreign keys. It is well known that the join of R_1 and R_2 on foreign key $R_1.A$ that references key $R_2.B$ has the same size of $R_1.A$, minus the number of tuples in R_1 such that A_1 is null. We assume that these are negligible, and therefore we estimate the size of $R_1 \bowtie_{A=B} R_2$ as the size of R_1.

(iii) At higher levels, selection nodes of the form $n' = \sigma_{A \in \{v_1,\ldots,v_l\}}(n)$ are introduced (note that we represent nodes at these levels lower in the lattice). These queries follow an interaction with the user, and a computation of the probability distribution for the values of attribute A in the result of query n. If l is equal to 1, the size of the selection query is equal to the frequency of value v_1. If $l > 1$, the size is equal to the sum of the frequencies of all v_is.

Based on this, for each lattice node n we determine the size of the sample, k_n, with the following two-step algorithm:

(a) We fix three configuration parameters, k_{min}, k_{max} and p_k. The first one is the minimum sample size – e.g., 1,000 tuples – the second one the maximum sample size – e.g., 100,000 tuples – the third the typical fraction of tuples to extract – e.g., 1%.

(b) We determine K_n by using the approach described above, then compute $K_n \times p_k$. If this is greater than k_{min} and lower than k_{max}, we return this as the desired value for k_n. Otherwise, we return k_{min} or k_{max}, respectively.

6.2.2. Order by Random – SAMPLINGMM-ORDERBY

Once the size k_n of the sample for node n is known, one possible way to extract a random sample from the result of a query Q is to use the $random()$ function in the order by clause, and extract only the first k_n tuples. This amounts to running the following SQL query:

```
SELECT * FROM n ORDER BY random() LIMIT k_n
```

Intuitively, the DBMS associates a random number with each tuple in the query result, and uses this number to select the first k_n tuples to be used as a sample. We call this first variant of the sampling method SAMPLINGMM-ORDERBY.

From the statistical viewpoint, this provides a good guarantee that the sample is chosen uniformly at random over the tuple set. Unfortunately, to perform the ordering the DBMS needs to scan the entire query result, and this might slow the computation.

6.2.3. Windowing – SAMPLINGMM-WINDOW

To speed up the computation, we may restrict the extraction of the sample to a fixed-size window of the tuple set. More specifically, we first fix a window size, w_n, as a multiple of the sample size, k_n, e.g., 10 times larger. Then, we fix a random offset, o_n and extract w_n tuples from the tuple-set. Finally, we use the random ordering strategy on the window only. We do this by using the following nested SQL query:

```
SELECT *
FROM ( SELECT * FROM n OFFSET o_n LIMIT w_n)
ORDER BY random()
LIMIT k_n
```

This query is faster than the one in Section 6.2.2 but it does not guarantee uniform tuple extraction from the tuple-set. In fact, from the statistical viewpoint the quality of the sample depends strongly on the actual window that is selected. Given attribute A_i, if the distribution of values for A_i within the window is significantly different from the one within the entire tuple-set (since data are not independent and identically distributed in the tuple-set), the sample analysis might lead to misleading results. As an example of this, consider table AcmeUser about users of the AcmeBand device, and the case in which by chance we select a window with a large set of consecutive tuples referring to people from the north region only.

6.3. *The* HYBRID *Algorithm*

To summarize, in Section 6.1, we have introduced a family of algorithms – PUREDB and PUREDB-H – that rely exclusively on the database engine to perform the computation. This will guarantee the best accuracy in the computation of probability distributions, but might be unacceptably slow with large databases.

Then, in Section 6.2, we have explored the use of sampling to compute distributions in main memory. Sampling, however, requires to deal with two important issues. *First*, as discussed in Section 6.2, extracting samples from the database is not a trivial task. We may either favour accuracy – as in SAMPLINGMM-ORDERBY– at the expense of most of the performance gain we expect, or favour performance – as in SAMPLINGMM-WINDOW– with the risk of sacrificing the guarantee about the statistical properties of the extracted samples. *Second*, sampling itself might not be enough to present results to the user. In fact, once a sample has been extracted, we use it to compute value distributions for attribute values. Being based on just a sample of tuples, these distributions represent estimates of the actual distribution of values within the entire tuple set. Notice that value distributions have two roles in our exploration paradigm: (i) they are used to discover interesting features by running the statistical tests discussed in the next section, e.d., "the types of activities by users"; (ii) once a relevant feature has been discovered, a few distinctive values – in some cases even the entire distribution – are shown to trigger the next step in the exploration, e.g., "52% running, 19% cycling".

We argue that the use of sampling-based estimates may be acceptable for the purpose of running statistical tests (step (i) above). On the contrary, with sampling the quality of the outputs shown to the user is usually too low (step (ii) above). This may be unacceptable, as discussed in Section 7.

To address these concerns, we now introduce a HYBRID algorithm that tries to combine accuracy and scalability. The hybrid algorithm is based on the following core ideas:

(i) Sampling is necessary for performance purposes, and distribution estimates can be effectively used for discovering relevant features.

(ii) The system needs to be very accurate in showing distributions for relevant features to users. Fortunately, these usually represent only a small fraction of the attributes in the lattice.

For these reasons, we introduce a two-step process defined as follows: (a) we sample the tuple-set to run statistical tests, and filter out uninteresting attributes; (b) we run one SQL query with a group-by clause for each relevant feature in order to derive the empirical distribution of values.

The algorithm is indeed hybrid, because it mixes main-memory computation with SQL queries in order to process distributions. In this approach, tuple-sets need to be processed multiple times, first to extract samples, and then to run group-by queries for relevant features. Consider, for example, node n with attributes $A_i, \ldots, A_h$. With HYBRID, node n is queried several times: (a) first, to extract a sample of tuples, and estimate probability distributions for $A_1, \ldots, A_h$; these are fed to the statistical hypothesis tests, to identify, e.g., A_2, A_3, A_7 as relevant features; (b) then, n is queried three more times, with the following queries:

```
SELECT A2, count(A2) FROM n GROUP BY A2
SELECT A3, count(A3) FROM n GROUP BY A3
SELECT A7, count(A7) FROM n GROUP BY A7
```

Recall that n can be a conjunctive query of arbitrary complexity, with multiple joins and selections. To avoid recomputing the query multiple times (e.g., 4 times in our example), we may find it useful to materialize its tuple-set as a temporary table $\mathcal{T}_n$, and then run the four queries on $\mathcal{T}_n$.

The materialization of the result of query n opens up to an alternative strategy to extract samples, discussed in the following section.

6.3.1. *Sampling with Random Tables*

We introduce a variant of the sampling methods discussed in Section 6.2 that assumes that the tuple-set to sample has been materialized, and each tuple has been assigned a unique integer id. The sampling method uses random tables, and we apply it in our hybrid algorithm.

More specifically, given a table $\mathcal{T}_n$ to sample, we assume to have an estimate of the size of $\mathcal{T}_n$, K_n, as discussed above, and intend to extract a sample of k_n tuples. Sampling with random tables works as follows.

We first materialize a table *randomIds*, containing exactly k_n random integers in the interval $\{1, \ldots, K_n\}$. Since k_n is usually small, this can be done quickly either by generating a number of inserts in main memory, or by a stored procedure within the DBSM.

Then, we join table *randomIds* with table $\mathcal{T}_n$ using tuple IDs, to select the needed tuples. The advantage of this approach over the ones discussed in Section 6.2 is quite evident: it extracts a random sample with good statistical properties, while at the same time it overcomes the need to scan and reorder the entire tuple-set.

We compare the performance and quality of the hybrid algorithm with the others introduced so far in Section 8.

7. Comparing Distributions

Name	Description	H0	H1	Assumption
Two-sided t-test	Detecting variations in the mean value	d and d' refer to data coming from two normal distributions with equal means and equal but unknown variances	Unequal means	Normality of the distribution.
Two-sided Wilcoxon rank sum test	Detecting variations in the median	d and d' refer to data coming from continuous distributions with equal medians	Unequal medians	Continuous distribution
Two-sample Kolmogorov-Smirnov test	Detecting variations in the probability density function	d and d' refer to data coming from the same continuous distributions	Different distributions	Continuous distributions
Two-sample Chi-square test	Detecting variations in the frequency distribution	d and d' refer to data coming from the same frequency distribution	Different frequency distributions	Categorical data
Entropy-based test	Detecting variations in the empirical entropy of two distributions	d and d' refer to data coming from the same frequency distribution	Entropy	Categorical data

Table 1: The STATTEST(d, d') tool: some statistical tests that are available within the INDIANA system.

Comparing the distributions d, d' through the STATTEST(d, d') operator is crucial to differentiate between *interesting* and *uninteresting* features. As mentioned above, STATTEST(d, d') relies on theoretically-grounded statistical hypothesis tests. These tests are statistical techniques able to assess whether a given hypothesis on data is true or not.

In our specific case, the goal of STATTEST(d, d') is to assess whether we have enough statistical confidence to claim that a statistical discrepancy exists between d and d'. We model this problem by defining a *null hypothesis*, denoted as H_0, assuming that d and d' are not statistically different (hence referring to the same distribution). On the contrary, the *alternative hypothesis*, denoted as H_1, assumes that d and d' represent two different distributions. The goal of the statistical test in STATTEST(d, d') is to analyse d and d' so as to make a decision about accepting H_0 (hence rejecting H_1) or rejecting H_0 (hence accepting H_1).

To achieve this goal a test statistic $\mathcal{T}$ is computed on d, d' and then transformed into a conditional probability τ through a suitably defined transformation $\mathcal{P}_T$, i.e.,

$$\tau = \mathcal{P}_T(d, d'). \tag{1}$$

$\mathcal{T}$ and $\mathcal{P}_T$ are defined according to the specific statistical test that is considered to compare d and d' (see Table 7 for examples of such tests), while τ denotes the *p-value* of the test measures the probability of obtaining a value of $\mathcal{T}$ at least as extreme as the one actually observed, provided that H_0 is true. Hence, smaller values of τ, would suggest to reject H_0 and accept H_1.

Within the INDIANA system the operator STATTEST(d, d') behaves as follows:

$$\text{STATTEST}(d, d') = \begin{cases} 0, \text{ if } \tau < \alpha \\ 1, \text{alternatively} \end{cases} \quad (2)$$

where α, which is a system parameter, represents the *Type I error* measuring the probability to reject H_0 even when it is true. This parameter is usually set to 0.01 or 0.05.

A wide range of statistical tests are available in the literature differing in the goal of the analysis and the required assumptions. Without any ambition of being exhaustive, we list in Table 7 some statistical hypothesis tests that can be useful within the INDIANA system. The choice of the specific test to be used is left to the designer or can be made available to the user during the interaction phase to drive the next exploration steps.

Within the INDIANA system we also developed a test STATTEST$_{ent}(d, d')$ for categorical data, with the aim of analysing differences in the *estimated entropy* of distributions d and d'. To achieve this goal, data in d and d' are further subsampled to compute mean and standard deviation of estimated entropies in the two distributions, i.e. μ_e and σ_e for d and μ'_e and σ'_e for d'. When an intersection between the two confidence intervals build as $\mu_e \pm \gamma\sigma_e$ and $\mu'_e \pm \gamma\sigma'_e$ exists (e.g., with $\gamma = 3$), there is no statistical significance that d and d' are different from the entropy point-of-view. Differently, when their intersection is empty, we have enough statistical confidence that d and d' are statistically different. Within the INDIANA system, the test STATTEST$_{ent}(d, d')$ could be used in categorical data in place of the *Two-sample Chi-square test* when the assumptions of the latter on data (e.g., on the number of samples for each category) do not hold.

We emphasize that STATTEST(d, d') not only compares distributions with one another, but also with other statistical properties of the data, such as the sample moments (e.g., mean, standard deviations) or the sample entropy.

In order to increase the efficiency, the STATTEST(d, d') could be run in an iterative modality as explained in [8] through the use of multiple hypothesis test. In this case, in order to avoid the problem of fake detections due to the confidence of hypothesis tests, we considered the Bonferroni correction.

8. Experimental Results

The exploration strategies described in the latter sections have been implemented in a working prototype written in Java. Here, we report our experiments about the accuracy and scalability of those strategies on both real and synthetic datasets. All tests have been executed on a Intel i7 machine with 2.5Ghz processor and 16GB of RAM under MacOSX. The DBMS was PostgreSQL 9.5.

Datasets. In the experimental evaluation we considered six different datasets, five of which are real while the other one is synthetic.

(a) Auditel: is a dataset containing real data of the audience measurement of Italian television, collected by an independent media agency during a period of 4 months in 2013. It has 37 tables with 16 foreign keys and 192 attributes and contains 44 million of tuples (`http://recsys.deib.polimi.it/?page_id=76`).

(b) Bus: is a real-world scenario used by Dallachiesa et al. [9] with 6 tables, 28 attributes, and 5 foreign keys. It contains 284K tuples.

(c) Census: it contains demographic and employment data extracted by the U.S. Census Bureau (`https://archive.ics.uci.edu/ml/datasets/Adult`). The dataset is composed by a single table with 12 attributes and 45K tuples.

(d) Hospital: is based on real data from the US Department of Health & Human Services (`http://www.hospitalcompare.hhs.gov`). We used a normalized version taken from [15]. It contains 3 tables with 2 foreign keys, and a total of 16 attributes. The dataset has 143K tuples.

(e) Movies: this dataset contains 1 million of anonymous ratings of 3,900 movies made by 6,040 users of a movie rating portal and collected by the GroupLens Research Group (`http://grouplens.org/`). It has 3 tables, 14 attributes and 2 foreign keys.

(f) AcmeBand is the synthetic database of our running example. It has 4 tables, 22 attributes and 3 foreign keys. In order to test the scalability of our system, we used the instance-generation tool TOXGENE [3] to generate three variants with different size, AcmeBand-1M, AcmeBand-10M, AcmeBand-100M and AcmeBand-1B, with 1, 10, 100, 1000

	ACME BAND 1M (68 MBytes)					
	TIME (ms)		FEATURE ACCURACY			OUTPUT ACCURACY
	Mean	Std.Dev.	Precision	Recall	F-Meas.	
PureDB	3042	19	1,000	1,000	1,000	1,000
PureDB-H	2254	34	1,000	0,750	0,857	0,930
SamplingMM-OrderBy	1779	40	1,000	0,833	0,909	0,778
SamplingMM-Window	1448	92	0,917	0,917	0,917	0,667
Hybrid	2237	47	0,923	1,000	0,960	1,000

	ACME BAND 10M (711 MBytes)					
	TIME (ms)		FEATURE ACCURACY			OUTPUT ACCURACY
	Mean	Std.Dev.	Precision	Recall	F-Meas.	
PureDB	26669	415	1,000	1,000	1,000	1,000
PureDB-H	14026	121	1,000	0,750	0,857	1,000
SamplingMM-OrderBy	28910	1203	0,917	0,917	0,917	0,778
SamplingMM-Window	10940	1608	0,833	0,833	0,833	0,667
Hybrid	16963	803	0,923	1,000	0,960	1,000

	ACME BAND 100M (7.3 GBytes)					
	TIME (ms)		FEATURE ACCURACY			OUTPUT ACCURACY
	Mean	Std.Dev.	Precision	Recall	F-Meas.	
PureDB	337567	21025	1,000	1,000	1,000	1,000
PureDB-H	180959	2393	1,000	0,692	0,818	0,889
SamplingMM-OrderBy	461791	6866	0,923	0,923	0,923	0,667
SamplingMM-Window	95759	12601	0,923	0,923	0,923	0,667
Hybrid	159349	2672	0,923	0,923	0,923	1,000

	ACME BAND 1B (76.7 GBytes)					
	TIME (ms)		FEATURE ACCURACY			OUTPUT ACCURACY
	Mean	Std.Dev.	Precision	Recall	F-Meas.	
PureDB	3762067	41025	1,000	1,000	1,000	1,000
PureDB-H	2517259	52393	1,000	0,692	0,818	0,889
SamplingMM-OrderBy	7053041	74015	0,940	0,923	0,931	0,667
SamplingMM-Window	752022	37894	0,940	0,850	0,893	0,667
Hybrid	1071403	32269	0,940	0,923	0,931	1,000

	BUS (9 MBytes)					
	TIME (ms)		FEATURE ACCURACY			OUTPUT ACCURACY
	Mean	Std.Dev.	Precision	Recall	F-Meas.	
PureDB	1571	30	1,000	1,000	1,000	1,000
PureDB-H	2027	10	1,000	0,278	0,435	0,400
SamplingMM-OrderBy	1078	12	1,000	0,667	0,800	0,615
SamplingMM-Window	898	47	0,929	0,722	0,813	0,692
Hybrid	1401	35	1,000	0,667	0,800	1,000

	HOSPITAL (7 MBytes)					
	TIME (ms)		FEATURE ACCURACY			OUTPUT ACCURACY
	Mean	Std.Dev.	Precision	Recall	F-Meas.	
PureDB	1197	33	1,000	1,000	1,000	1,000
PureDB-H	2353	34	1,000	0,143	0,250	0,000
SamplingMM-OrderBy	741	34	0,800	0,571	0,667	1,000
SamplingMM-Window	536	36	0,800	0,571	0,667	0,800
Hybrid	1144	17	0,800	0,571	0,667	1,000

	MOVIES (21 MBytes)					
	TIME (ms)		FEATURE ACCURACY			OUTPUT ACCURACY
	Mean	Std.Dev.	Precision	Recall	F-Meas.	
PureDB	4748	89	1,000	1,000	1,000	1,000
PureDB-H	3235	55	1,000	0,462	0,632	1,000
SamplingMM-OrderBy	2764	120	0,923	0,923	0,923	0,818
SamplingMM-Window	1688	272	0,917	0,846	0,880	0,455
Hybrid	3794	147	0,929	1,000	0,963	1,000

	AUDITEL (1.1 GBytes)					
	TIME (ms)		FEATURE ACCURACY			OUTPUT ACCURACY
	Mean	Std.Dev.	Precision	Recall	F-Meas.	
PureDB	195651	1329	1,000	1,000	1,000	1,000
PureDB-H	61141	3827	0,707	0,446	0,547	0,930
SamplingMM-OrderBy	184156	7074	0,974	0,804	0,881	0,812
SamplingMM-Window	37489	2875	0,706	0,652	0,678	0,465
Hybrid	94490	6717	0,961	0,804	0,876	1,000

Table 2: Full results of the experimental evaluation

millions of tuples, respectively. We configured the data generator in such a way as to control the distributions of values within instances, and therefore the number of interesting features.

Database sizes in bytes are reported in Table 2. It is possible to notice that some of the database are very large in size, and would hardly fit in main memory.

Experiments. To evaluate the proposed system we conducted the following five main experiments:

(i) **Execution Times and Scalability**: in the first experiment we run our system on various datasets to measure execution times (in ms) and scalability wrt to large databases.

(ii) **Quality and Accuracy**: then, we compare the algorithms introduced in the paper in terms of accuracy, i.e.: (a) precision and recall in identifying relevant features; (b) quality of the outputs that are presented to users. Notice that algorithm PUREDB is guaranteed to have both precision and recall equal to 1 in identifying relevant features, since it considers the entire tuple set when computing empirical distributions. As a consequence, in the following we study the quality of the other algorithms in comparison to PUREDB.

(iii) **Sampling**: we study how quality and scalability vary with different sample sizes.

(iv) **Comparison to SeeDB**: We compare INDIANA to the SeeDB visualization recommender system, and report on the respective quality of the suggestions.

(v) **User Study**: finally, we conduct a user study in order to measure how effective final users consider this approach.

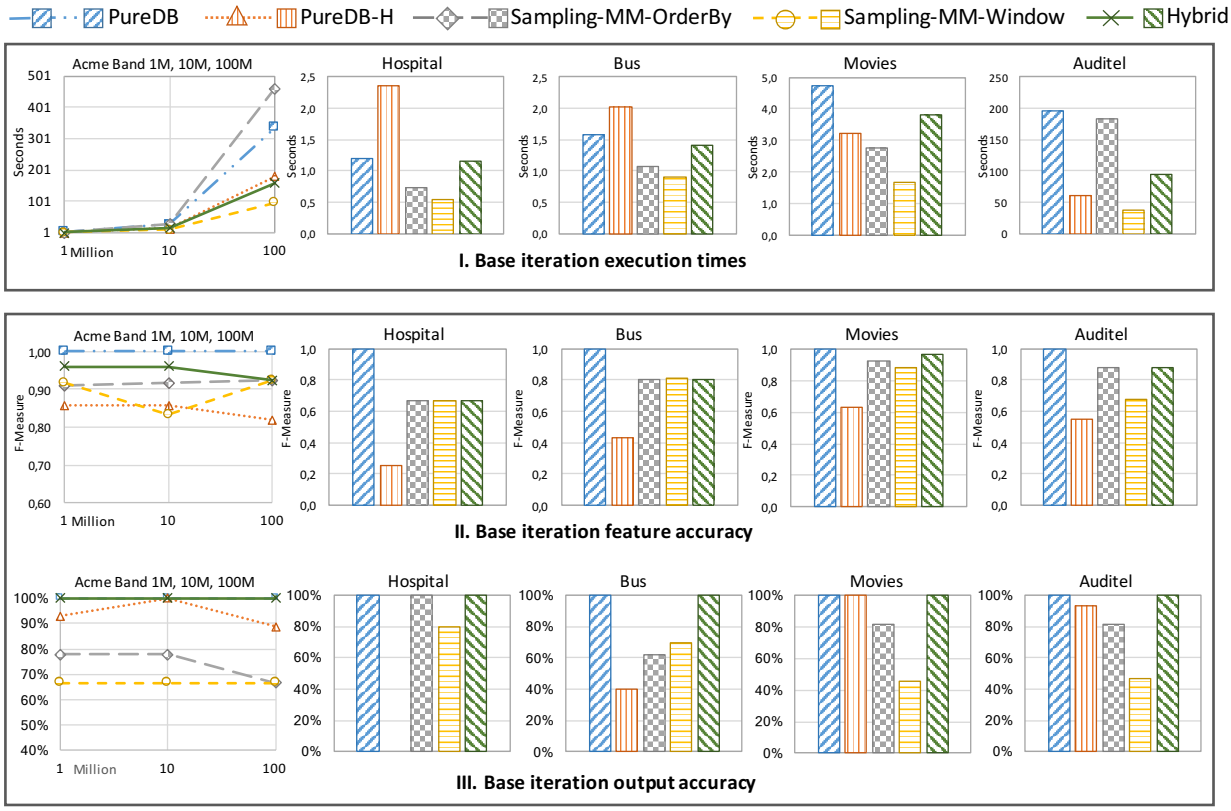


Figure 3: A comparison among different feature discovery algorithms

8.1. Execution Times and Scalability

The goal of this set of experiments is to compare the scalability of the five variants of the discovery algorithm described in the paper, namely PUREDB, PUREDB-H, SAMPLINGMM-ORDERBY, SAMPLINGMM-WINDOW and HYBRID. In order to do that we focus on the base iteration, since, as discussed in Section 6, it is the crucial step of the entire process. Recall that the base iteration requires to generate nodes at level 0 and 1, and then to analyze their attributes in order to identify relevant features to show to users (see Algorithm 1).

Each experiment has been executed five times and we report the average execution time in Figure 3.I. The full results of the experiments are reported in Table 2. For the variants that use sampling, we determine the size of the sample as described in Section 6.2.1, with k_{min} = 1,000, k_{max} = 100,000 and p_k = 1%.

Let us first discuss the performance of the PUREDB algorithm. It can be seen (Figure 3.I) that on small scenarios (Hospital and Bus) there are no significant differences in execution times between PUREDB and sampling-based methods. This is somehow expected, since sampling introduces an overhead that may become evident when full result-sets are not very large. This is even more evident in PUREDB-H, where the cost of updating the database statistics is higher than the time needed to execute the group-by queries. However, in scenarios with larger databases, like Auditel and AcmeBand-100M, sampling significantly outperforms PUREDB whose execution times are somehow inadequate for an interactive tool (iteration one with PUREDB requires more than 5 minutes on AcmeBand-100M).

SAMPLINGMM-ORDERBY– that uses queries with 'order by $random()$' clauses to perform the sampling – was also unsatisfactory. This strongly depend on the size of the result-set to sample, since the DBMS needs to scan it entirely. On Auditel and AcmeBand-100M, in fact, SAMPLINGMM-ORDERBY is almost as slow as PUREDB.

The SAMPLINGMM-WINDOW algorithm consistently achieved the best execution times, since it only samples a small window of tuples over the result-set.

Performance was good also with the HYBRID variant. Recall that this differs from SAMPLINGMM in two main aspects. First of all, the random extraction is performed by generating a table containing a random sample of tuple

ids. In this way, the performance is not affected by the size of the original data, but depends only on the size of the sample, which is usually very small. The second difference is that HYBRID does not rely on the distribution that was estimated using the sample; rather, it executes a final group-by query for each interesting feature to find out the actual distribution. As a consequence, it is slightly slower than SAMPLINGMM on smaller scenarios. As a counterpart, these two differences guarantee better accuracy, as we will show in the next experiment.

8.2. Quality and Accuracy

There are two main quality factors that are related to the interaction between system and user:

(a) precision and recall in identifying the relevant features contained in the data; we call this *feature accuracy*;

(b) correctness of the distributions that are shown to users as part of the output for relevant features; we call this *output accuracy*.

Recall that algorithm PUREDB, by construction, has 100% quality and accuracy. As a consequence, to measure the feature accuracy for the other algorithms, for each dataset we fixed the expected set of relevant features as the result of the the PUREDB algorithm, that is guaranteed to have perfect precision and perfect recall. For the AcmeBand datasets we have 12 relevant features in the first two steps, 92 for Auditel, 18 for Bus, 7 for Hospital and 13 for Movies. Then, we compare this ground truth to the set of features identified by each algorithm, and measure precision, recall and f-measure. F-Measures are reported in Table 2.

We first note that PUREDB-H often fails to identify the correct features. This confirms the idea that the histograms collected by the DBMS and stored in the database catalog are too inaccurate to be used in our application.

Among the sampling strategies, SAMPLINGMM-WINDOW was usually the one with the worst results. In fact, its outputs strongly depend on the selected window.

On the positive side, SAMPLINGMM-ORDERBY and HYBRID have essentially the same feature accuracy – since they are two different implementations of the same sampling strategy – and this is often quite high (from 80% to 96%). Only in one case, namely on the Hospital dataset, the feature accuracy is lower than 70%. This is due to the fact that this dataset contains very few relevant features, and the distributions of values within the relevant features are quite close to being uniform.

We next discuss output accuracy. To measure the distance between the actual distribution of values and the ones that were estimated by the different algorithms, we used the two-sample Chi-square and measured the number of cases in which the test was successful. For example, when considering the Auditel dataset, only 61 out of the 74 relevant features discovered by the SAMPLINGMM-ORDERBY algorithm have been considered as similar to the actual distribution. This means that the algorithm correctly identifies the remaining 13, but shows misleading outputs to users. The output accuracy is 82% in this case.

Our tests confirm that, while sampling can be effective in identifying the relevant features within the database, it may lead to misleading outputs, i.e., it might show relative frequencies of values that are often quite different from the actual one.

The HYBRID algorithm, on the contrary, is not affected by this limitation. In fact, as expected, it outperformed all other sampling strategies in terms of output accuracy. We can conclude that HYBRID represents the best tradeoff between quality and scalability.

8.3. Impact of Sample Size

We now want to study the impact of the size of samples on quality and scalability. In order to do that, we fix two scenarios (Auditel and Movies) and measure execution times and feature accuracy for the HYBRID algorithm using samples of increasing sizes, varying from 0,1% to 10%. The results are reported in Figure 4 (in logaritmic scale for execution times).

As expected, increasing the size of samples improves accuracy. However, it is interesting to note that quality grows more slowly than execution times. So it is crucial to find a good trade-off between accuracy and scalability. A natural breaking point can be the time needed to run the PUREDB algorithm, that is guaranteed to have perfect accuracy. Our experiments show that, on Auditel and Movies, this corresponds to sample sizes between 1% and 3%, but it is possible to achieve comparable accuracy with better execution times using samples in the 0.5%-1% range.

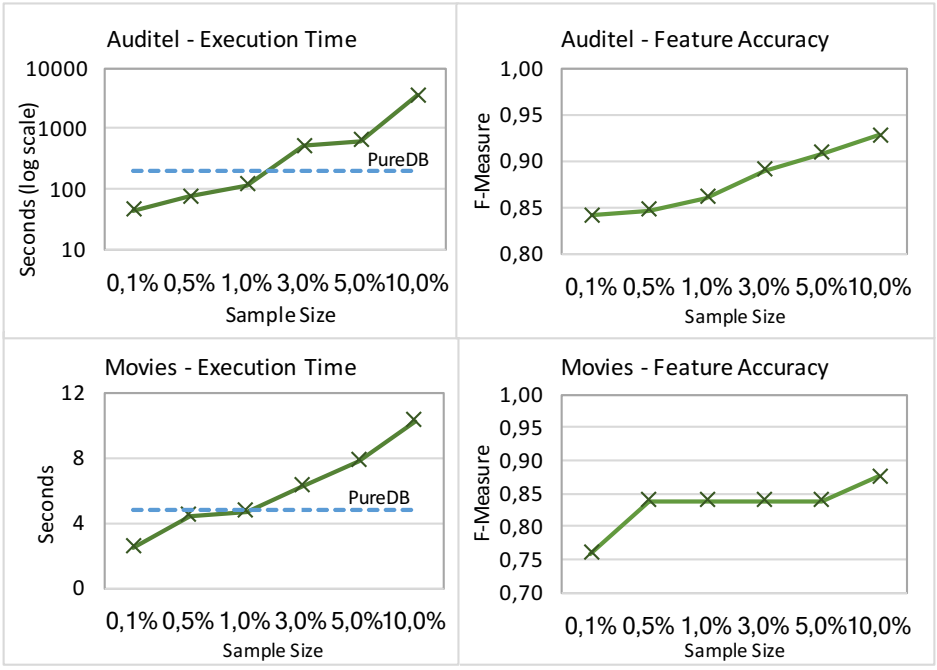


Figure 4: Execution times and feature accuracies of the HYBRID algorithm on Auditel and Movies with different sample sizes

8.4. Comparison with SEEDB

In this section we compare the quality of the suggestions proposed by INDIANA to those provided by SEEDB [31]. SEEDB is a visualization recommendation tool that helps users to find interesting visualizations inside a relational database. In particular, given two different queries (e.g., adults vs married adults) on a database, it returns the top-k most interesting aggregate views (e.g. the maximum age of male married adults is significantly higher than those of unmarried adults). An aggregate view is a group-by query, identified by a dimension attribute (the attribute to group-by, in our example the sex), a measure attribute (the attribute to aggregate, in our example, the age) and an aggregation function (max in our example). They define the concept of interestingness of an aggregate view based on the differences between the probability distributions resulting from executing the view on the two base queries.

Since INDIANA and SEEDB share several aspects, this section aims at comparing their performance. However, there are several important observations we need to make about this comparison. First, the two systems have different inspirations and can be compared only to a certain extent.

In essence, SEEDB is conceived to work in a rather specific setting – it analyzes a single table at a time, and it requires the user to explicitly trigger the analysis by selecting two queries over the database – but it is very expressive in terms of aggregate functions that can be discovered. Its goal is essentially to discover relevant statistical deviations between the results of the two input queries.

On the contrary, INDIANA is conceived to be a fully-automatic tool for supporting complex database explorations, with no prior configuration or input from the user. It supports exploration paths involving the base tables, and complex views that are progressively built from them, also using joins and selections. As a counterpart, it has a more focused notion of *relevance* for attributes, based on their value histograms.

As a consequence, our purpose is not to compare the actual user experience or the overall effectiveness of the two systems. We rather want to conduct a more focused comparison of the recommendations that are generated by the two systems, with the primary goal of assessing the effectiveness of the recommendations provided by INDIANA using SEEDB as a baseline. With the purpose of reporting a fair comparison, we designed our experiment as follows:

(i) first of all we manually selected three different explorations over the Census database. Each exploration is composed by two queries (as shown in Table 8.4).

(ii) Then we asked 6 data-analysis experts to compare the result of query 1 and query 2 in order to list a number of

interesting attributes, i.e., features. To do that, we presented to each expert the value distributions of the 12 attributes for the two queries, and we asked them to rate each attribute on a scale of *interesting*, *neutral* and *not interesting*.

(iii) For each exploration, we identified a set of *ranked interesting features*. To do this, we assigned a score to each attribute as follows: (i) one point whenever an expert had considered it as interesting; (ii) no points whenever an expert had considered it as neutral; (iii) minus one point whenever an expert had considered it as not interesting. Attributes were ranked according to their score, and an attribute was deemed as interesting if it had a score of at least 1 (i.e., the positive votes of the expert were more than the negative ones). The set of interesting features ranked according to their scores were selected as the *ground truth* that we want to achieve. We call these features $feat_{ground}$. The number of interesting features for the three explorations are respectively 5, 6, and 5. Full results are reported in Figure 5. The table reports for each exploration, each expert and each attribute in the database, whether the expert found that attribute to be interesting. Notice how, in some of the explorations one or more attributes had a fixed value – e.g., attribute **A5** in Exploration 1. For those we did not collect scores from the experts. The table shows a very good level of agreement among the experts, in the sense that their scores tend to overlap.

Then, we ran the two systems on the three different explorations and compared the results. To make the results comparable to each other, we made the following assumptions:

(a) we disabled the exploration module of INDIANA and fixed the nodes of the lattice to compare; in essence, we limited the system to running step (ii) of the feature discovery algorithm in Section 5.

(b) to run SEEDB we used a public implementation available online (`https://github.com/rahmansunny071/seedb`). We asked SEEDB to find the most interesting aggregate views for the two input queries, with the following parameters. (i) The aggregate function used is `COUNT`. (ii) The aggregate attribute has a uniform distribution. In essence, these parameters limit the scope of the view-selection algorithm in SEEDB to searching only for the attribute to group-by, in a way that is more similar to what INDIANA does. The results are finally sorted by distance in descending order. We used the default distance function (Earth Mover's Distance) to compare the normalized distributions, as suggested in the SEEDB paper.

As a result, we had two ranked lists of attributes, which we denote by $feat_{\text{INDIANA}}$ and $feat_{\text{SEEDB}}$, respectively. These were compared to $feat_{ground}$ using the normalized discounted cumulative gain [32] measure. The results are shown in Figure 6.

We observe that in all of three explorations the quality of the suggestions provided by INDIANA is higher than that for SEEDB, and in two cases is above 0.95. In essence, INDIANA was able to capture what the human experts perceived as "interesting features" of the data at hand better than SEEDB did. We believe that these results confirm the more robust nature of the feature-selection algorithm implemented in INDIANA, and that leveraging statistical-hypothesis tests actually reduce the risk of false discoveries.

Notice that there is a significant difference between the two systems also in the number of results that were returned. In fact, the number of suggested features returned by INDIANA for the three explorations are respectively 7, 6 and 4. These features are very close to the ones selected on the basis of the human expert ratings. We report in Figure 7 the F-Measure computed between $feat_{ground}$ and $feat_{\text{INDIANA}}$. More specifically:

(i) For the first exploration our system returns two extra features. However the Hellinger distance for both of them is very low, so these features receive a very low ranking from the system.

(ii) For the second exploration we have a perfect match between $feat_{\text{INDIANA}}$ and $feat_{\text{SEEDB}}$.

(iii) The third exploration is the only one in which INDIANA was unable to identify a feature marked as interesting by the experts.

On the contrary, we cannot compare our results to the ones generated by SEEDB in terms of F-measure, since SEEDB returned all of the 12 attributes in all explorations. Notice that the system has a pruning module that can efficiently select the top-K visualizations, but the value of K needs to be specified by the user.

8.5. User Study

We conducted a comparative experiment to prove the effectiveness of INDIANA to final users. We discuss the results in this section.

Exploration Tools To compare INDIANA to a baseline, we implemented another data-exploration system that we called BELLOQ. BELLOQ allows users to explore a database by graphically formulating queries and inspecting at-

Name	Query 1	Query 2
Exp1	All adults	Adults who earn more than 50K/year
Exp2	All adults	Adults who work less than 30 hours/week
Exp3	Unmarried adults	Unmarried adults who earn more than 50K/year

Table 3: Three different explorations over Census db

Exploration 1

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12
Expert 1	1	-1	-1	1	-	1	1	-1	1	-1	1	-1
Expert 2	1	-1	-1	1	-	1	1	0	-1	-1	-1	-1
Expert 3	1	-1	-1	1	-	0	1	-1	1	-1	-1	-1
Expert 4	1	-1	-1	1	-	1	1	-1	1	0	0	0
Expert 5	1	0	0	1	-	1	0	0	1	-1	1	1
Expert 6	1	-1	-1	-1	-	-1	1	-1	0	0	0	-1
Result	**6**	**-5**	**-5**	**4**	**-**	**3**	**5**	**-4**	**3**	**-4**	**0**	**-3**

Exploration 2

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12
Expert 1	1	-1	-1	0	1	-	0	-1	1	-1	1	-1
Expert 2	1	-1	-1	1	1	-	0	-1	1	-1	0	-1
Expert 3	1	-1	-1	0	1	-	0	0	0	0	1	-1
Expert 4	1	-1	-1	1	0	-	1	-1	0	-1	-1	0
Expert 5	1	-1	-1	0	1	-	1	0	1	-1	1	1
Expert 6	1	-1	-1	0	1	-	1	-1	1	-1	0	-1
Result	**6**	**-6**	**-6**	**2**	**5**	**-**	**3**	**-4**	**4**	**-5**	**2**	**-3**

Exploration 3

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12
Expert 1	1	-1	-1	1	-	1	-	-1	1	-1	1	-1
Expert 2	1	-1	-1	1	-	0	-	-1	-1	-1	1	-1
Expert 3	1	-1	-1	1	-	0	-	0	-1	0	0	0
Expert 4	1	-1	-1	1	-	1	-	-1	1	-1	0	0
Expert 5	1	-1	0	0	-	1	-	0	1	0	1	1
Expert 6	1	-1	-1	1	-	1	-	-1	1	-1	1	1
Result	**6**	**-6**	**-5**	**5**	**-**	**4**	**-**	**-4**	**2**	**-4**	**4**	**0**

A1: age, A2: capital gain, A3: capital loss, A4: education, A5: income category, A6: hours per week, A7: married, A8: native country, A9: occupation, A10: race, A11: sex, A12: workclass

Figure 5: Score assigned by each expert to attributes in the three explorations of Census db. Red attributes are the ones classified as interesting.

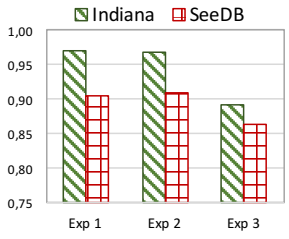


Figure 6: The quality of the suggestions provided by INDIANA and SEEDB for the Census db, in terms of normalized discounted cumulative gain.

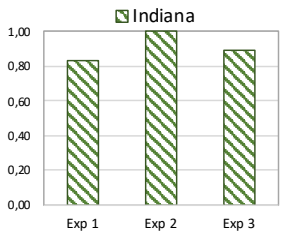


Figure 7: The quality of the suggestions provided by INDIANA for the Census db, in terms of F-Measure.

tribute distributions. More specifically, BELLOQ offers a GUI to perform the following actions: (i) select one or more tables to explore; multiple tables are automatically joined based on referential-integrity constraints; (ii) select one or more attributes, and specify selections over them; joins and selections compose a query over the db; (iii) explore the value distributions of attributes in the query result. For example s/he may want to explore the join of tables AcmeUsers and Activity with a selection *type=running*. In its essence, BELLOQ offers the same exploration features provided by INDIANA, without providing any suggestions; as a consequence, it forces users to manually design their

INDIANA

	Explored		Evaluated		Like		Dislike			
	Total	Avg	Total	Avg	Total	Avg	Total	Avg	Diff	Ratio
MOVIES	717	32,6	345	15,7	253	11,5	92	4,2	77%	68%
AUDITEL	566	25,7	249	11,3	178	8,1	71	3,2	77%	59%

BELLOQ

	Explored		Evaluated		Like		Dislike			
	Total	Avg	Total	Avg	Total	Avg	Total	Avg	Diff	Ratio
MOVIES	151	6,9	129	5,9	69	3,1	60	2,7	50%	23%
AUDITEL	120	5,5	81	3,7	32	1,5	49	2,2	27%	5%

Table 4: User Study: Feature Exploration Results

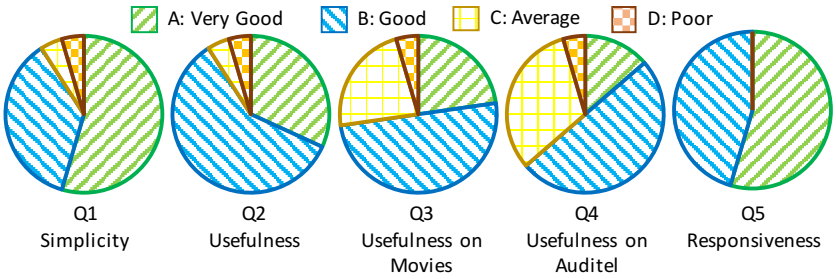


Figure 8: User Study: Pool Results

own exploration paths.

Test Setup The study involved 44 undergraduate students in computer science. Students were assigned two tasks that required to explore a database.

- task 1 consisted of writing a newspaper article about the internet movie databases, based on the content of the Movies db, or tv audience in Italy, based on the content of the Auditel db; users were left free to choose the topics to write about, as long as they were based on "interesting" evidence collected from the databases;
- task 2 consisted of suggesting marketing decisions, i.e., suggesting advertising campaigns for targeted users, either based on movie ratings, or on television audience; also in this case, users were free to pick up both the object to advertise, and the category of users, based on their tv watching habits.

We divided the students in two disjoint groups, and we assign, to each group one of the two tools in analysis, either INDIANA or BELLOQ.

We trained the students with a 20-minute tutorial based on the (synthetic) AcmeBand dataset. Then, we asked them to explore both the Movies and Auditel datasets for the purpose of completing the tasks we had assigned them. They had 20 minutes for each dataset. We collected two main feedbacks about explorations, as discussed in the following paragraphs.

Feature Exploration Feedback The INDIANA and BELLOQ GUIs offer *like* and *dislike* buttons for features. Students were instructed to use the buttons to provide feedbacks about how "interesting" they felt a feature was with respect to

	A	B	C	D
Q1: SIMPLICITY	55%	36%	5%	5%
Q2: USEFULNESS	32%	59%	5%	5%
Q3: USEFULNESS ON MOVIES	23%	50%	23%	5%
Q4: USEFULNESS ON AUDITEL	14%	50%	32%	5%
Q5: RESPONSIVENESS	55%	45%	0%	0%

Table 5: User Study: Pool Results Table

the task they needed to carry on. Results about likes and dislikes are reported in Table 4 We notice that, when using INDIANA, users were able to explore a significant portion of the database in the assigned 20: on the Movies dataset each student explored on average 33 features, while on the Auditel dataset they were 26. On the contray, BELLOQ users covered on average a much smaller number of features – 6.9 and 5.5 respectively. This proves the effectiveness of the paradigm provided by INDIANA in terms of exploration times. In general, the Auditel dataset revealed to be more difficult to explore, due to the larger number of features, and due to the fact that some of the attributes do not have a clear semantics (e.g., attribute "lag" in table "Channel" with values 0, 1, 2, 24 is used to refer to those channels, like "Sky Movies+24" that air a show 24 hours after it has been shown on "Sky Movies").

We were able to collect many feedbacks about the features explored by the users. Overall, with INDIANA, users provided feedbacks about approximately 50% of the features they explored. Notice that these percentages are significantly higher with BELLOQ. We consider this a consequence of the increased effort required by BELLOQ to inspect a feature. On Movies, INDIANA users on average liked 12 of the features they had explored, and disliked 4. A similar behavior was observed on Auditel, with an average of 11 liked features, and 3 that were disliked. For 77% of the users, the number of likes was higher than the number of dislikes (see column *Diff* in Figure 4), and in most cases the number of likes was two-time larger than the one of dislikes (column *Ratio*). On the contrary, with BELLOQ these numbers are lower – 50% and 23% on Movies, 27% and 5% on Auditel.

We believe that these results confirm that, in complex exploration tasks, the availability of a tool that provides good-quality suggestions is crucial in order to guide users and enhance productivity.

Poll Then, we asked users to answer a 5-question poll in which we asked to rate the system on a scale of A=very good, B=good, C=adequate, D=poor in terms of: $(Q1)$ Overall simplicity $(Q2)$ Overall usefulness $(Q3)$ Usefulness in exploring the Movies dataset $(Q4)$ Usefulness in exploring the Auditel dataset $(Q5)$ Response times.

Results are summarized in Figure 8 and Table 5. All users felt that the system was either very responsive (55%) or responsive (45%). Also, a vast majority of users considered the overall simplicity of the system to be very good (55%) or good (36%). The overall usefulness of the system was rated as very good by 32% of the users, and good by 59%. Users gave mixed response for the Auditel dataset, more than they did for Movies due to the intrinsic complexity of the database itself.

9. Conclusions

We have introduced INDIANA, a system to assist the database exploration through a novel interactive process between system and user. Our experimental results confirm the effectiveness and efficiency of the proposed solution.

We believe that INDIANA represents a very promising starting point to further investigate the issue of exploring large datasets. A prominent research direction is related to the notion of relevance: we believe that extensive user tests in a crowdsourcing environment could be used to refine our statistical notion of relevance based on the subjective perception of users. Another important extension is related to the development of advanced visualization paradigms that may help to communicate to users the distribution of data and the relevance of features.

References

[1] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Eighth Eurosys Conference 2013*, pages 29–42, 2013. doi: 10.1145/2465351.2465355. URL http://doi.acm.org/10.1145/2465351.2465355.

[2] E. Baralis, P. Garza, E. Quintarelli, and L. Tanca. Answering XML queries by means of data summaries. *TODS*, 25(3), 2007. doi: 10.1145/1247715.1247716. URL http://doi.acm.org/10.1145/1247715.1247716.

[3] D. Barbosa, A. O. Mendelzon, J. Keenleyside, and K. A. Lyons. Toxgene: a template-based data generator for XML. In *Proc. of ACM SIGMOD International Conference on Management of Data*, page 616, 2002. doi: 10.1145/564691.564769. URL http://doi.acm.org/10.1145/564691.564769.

[4] L. Battle, M. Stonebraker, and R. Chang. Dynamic reduction of query result sets for interactive visualizaton. In *Proc. of the 2013 IEEE International Conference on Big Data*, pages 1–8. IEEE, 2013.

[5] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1363–1375, 2016.

[6] C. Binnig, L. D. Stefani, T. Kraska, E. Upfal, E. Zgraggen, and Z. Zhao. Toward sustainable insights, or why polygamy is bad for you. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research*, 2017.

[7] N. D. Blas, M. Mazuran, P. Paolini, E. Quintarelli, and L. Tanca. Exploratory computing: a challenge for visual interaction. In *AVI*, pages 361–362, 2014. doi: 10.1145/2598153.2600037. URL http://doi.acm.org/10.1145/2598153.2600037.

[8] M. Buoncristiano, G. Mecca, E. Quintarelli, M. Roveri, D. Santoro, and L. Tanca. Database challenges for exploratory computing. *SIGMOD Rec.*, 44(2):17–22, Aug. 2015. ISSN 0163-5808. doi: 10.1145/2814710.2814714. URL http://www.db.unibas.it/users/santoro/articles/18.SIGMOD_REC2015.pdf.

[9] M. Dallachiesa, A. Ebaid, A. Eldawy, A. K. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. NADEEF: a Commodity Data Cleaning System. In *SIGMOD*, pages 541–552, 2013.

[10] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. M. Lohman. Dynamic faceted search for discovery-driven analysis. In *Proc. of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008*, pages 3–12, 2008. doi: 10.1145/1458082.1458087. URL http://doi.acm.org/10.1145/1458082.1458087.

[11] Ç. Demiralp, P. J. Haas, S. Parthasarathy, and T. Pedapati. Foresight: Recommending visual insights. *PVLDB*, 10(12):1937–1940, 2017.

[12] K. Dhamdhere, K. S. McCurley, R. Nahmias, M. Sundararajan, and Q. Yan. Analyza: Exploring data with conversation. In *Proceedings of the 22Nd International Conference on Intelligent User Interfaces*, IUI '17, pages 493–504. ACM, 2017. ISBN 978-1-4503-4348-0.

[13] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: an automatic query steering framework for interactive data exploration. In *International Conference on Management of Data, SIGMOD 2014*, pages 517–528, 2014.

[14] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh. Querie: Collaborative database exploration. *IEEE Transactions on Knowledge & Data Engineering*, 26(7):1778–1790, 2014. ISSN 1041-4347.

[15] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. Mapping and cleaning. In *Proceedings of the 30th International Conference on Data Engineering*, ICDE 2014, pages 232–243. IEEE, Mar. 2014. ISBN 978-1-4799-3480-5. doi: 10.1109/ICDE.2014.6816654. URL http://www.db.unibas.it/users/santoro/articles/12.ICDE2014.pdf.

[16] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *PVLDB*, pages 466–475, 1997. URL http://www.vldb.org/conf/1997/P466.PDF.

[17] P. Hanrahan. Analytic database technologies for a new kind of user: the data enthusiast. In *SIGMOD*, pages 577–578, 2012. doi: 10.1145/2213836.2213902. URL http://doi.acm.org/10.1145/2213836.2213902.

[18] F. Herrera, C. J. Carmona, P. González, and M. J. del Jesús. An overview on subgroup discovery: foundations and applications. *Knowl. Inf. Syst.*, 29(3):495–525, 2011. doi: 10.1007/s10115-010-0356-2. URL http://dx.doi.org/10.1007/s10115-010-0356-2.

[19] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *Proc. of the 2015 ACM SIGMOD*, pages 277–281, 2015.

[20] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *Proc. of the ACM SIGMOD International Conference on Management of Data, 2007*, pages 13–24, 2007. doi: 10.1145/1247480.1247483. URL http://doi.acm.org/10.1145/1247480.1247483.

[21] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and interactive cube exploration. In *IEEE 30th International Conference on Data Engineering*, pages 472–483, 2014.

[22] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The researcher's guide to the data deluge: Querying a scientific database in just a few seconds. *PVLDB*, 4(12):1474–1477, 2011. URL http://www.vldb.org/pvldb/vol4/p1474-kersten.pdf.

[23] A. Kim, E. Blais, A. G. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 8(5):521–532, 2015.

[24] W. Klösgen. Explora: A multipattern and multistrategy discovery assistant. In *Advances in Knowledge Discovery and Data Mining*, pages 249–271. 1996.

[25] K. Morton, M. Balazinska, D. Grossman, and J. D. Mackinlay. Support the data enthusiast: Challenges for next-generation data-analysis systems. *PVLDB*, 7(6):453–456, 2014. URL http://www.vldb.org/pvldb/vol7/p453-morton.pdf.

[26] V. Poosala, V. Ganti, and Y. E. Ioannidis. Approximate query answering using histograms. *IEEE Data Eng. Bull.*, 22(4):5–14, 1999. URL http://sites.computer.org/debull/99dec/poosala.ps.

[27] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. In *Advances in Database Technology - EDBT'98*, pages 168–182, 1998. doi: 10.1007/BFb0100984. URL http://dx.doi.org/10.1007/BFb0100984.

[28] F. Tauheed, T. Heinis, F. Schürmann, H. Markram, and A. Ailamaki. SCOUT: prefetching for latent feature following queries. *PVLDB*, 5 (11):1531–1542, 2012.

[29] J. W. Tukey. *Exploratory data analysis*. Reading, Mass.

[30] D. Tunkelang. *Faceted Search*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2009. doi: 10.2200/S00190ED1V01Y200904ICR005. URL http://dx.doi.org/10.2200/S00190ED1V01Y200904ICR005.

[31] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: Efficient data-driven visualization recommendations to support visual analytics. *Proc. VLDB Endow.*, 8(13):2182–2193, Sept. 2015.

[32] Y. Wang, L. Wang, Y. Li, D. He, W. Chen, and T.-Y. Liu. A Theoretical Analysis of Normalized Discounted Cumulative Gain (NDCG) Ranking Measures. In *Proceedings of the 26th Annual Conference on Learning Theory (COLT)*, 2013.

[33] K. Wongsuphasawat, D. Moritz, A. Anand, J. D. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Trans. Vis. Comput. Graph.*, 22(1):649–658, 2016.

[34] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. D. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2648–2659, 2017.