open proceedings

DBExplorer: Exploratory Search in Databases

Manish Singh
Indian Institute of Technology
Hyderabad, India
msingh@iith.ac.in

Michael J. Cafarella
University of Michigan
Ann Arbor, USA
michjc@umich.edu

H.V. Jagadish
University of Michigan
Ann Arbor, USA
jag@umich.edu

ABSTRACT

A traditional relational database can evaluate complex queries but requires users to precisely express their information need. But users often do not know what information is available in a database, and hence cannot correctly express their information need. Traditional databases do not provide convenient means for users to gain familiarity with the data.

In this paper, we study the problem of exploratory search, which a user may wish to perform to get an understanding of the data set. We note that users often have some decisions already made, so what they need is not an overall database summary, but rather a summary "in context" of the relevant portion of the database. Towards this end, we devise a novel data summarization technique called the Conditional Attribute Dependency (CAD) View, which shows the conditional dependencies between attribute values conditioned on applied selections. The CAD View can help users to gain familiarity with structured datasets in an attribute-wise manner.

To evaluate the CAD View, we perform a user study comprising three complex exploratory tasks on a real dataset. Our studies show that users are able to do all the tasks about 4-5 times faster and with better accuracy using the CAD View compared to the data summary shown in faceted navigation, which is currently the most popular search interface for e-commerce and has support for exploratory search.

1. INTRODUCTION

Users today have access to many large databases, yet find it difficult to access the records they want. In some cases, the challenge is to write correct SQL. But databases today often come with easy-to-use query interfaces. Users still find it difficult to specify the precise query conditions, due to limited familiarity with the data. Consider, for example, a user on a travel web site looking to book a hotel in a big city. If she knows her preferences for price, location, star rating, and other such relevant attributes, she can easily specify a query that will pull out a few good choices for her to consider from among the hundreds of hotels in the city. But, if she is unfamiliar with the city, she may not understand what typical prices are in the city or how all the 5-star hotels are clustered in the financial district or how there is a tradeoff between location and price. Without this knowledge of the data in the database, she is forced to depend on other data sources, such as advice from friends and relatives, social media, web documents, etc., to gain data familiarity and pose the right queries. In consequence, even after hours of effort she may be left with various doubts: "Did I make a good choice?" "Did I explore all my options?" "Did I spend more than I needed to?"

©2016, Copyright is with the authors. Published in Proc. 19th International Conference on Extending Database Technology (EDBT), March 15-18, 2016 - Bordeaux, France: ISBN 978-3-89318-070-7, on OpenProceedings.org. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

Our goal in this paper is to develop database facilities to support exploratory search. There are two types of search: *lookup* and *exploratory* [26, 24, 19]. In lookup search users have a specific well-defined search goal. In contrast, in exploratory search, the users' goal is to gain a comprehensive understanding of data that will enable them to pose more informed lookup queries.

Supporting data exploration is difficult because: (a) Datasets are complex and heterogeneous, and (b) Users have diverse needs. It is easy, for example, to provide the user with some simple summary statistics, such as average price for a hotel room. However, this number is of only limited value to the user, perhaps because there is huge variance between different parts of the city or perhaps because the user is a backpacker looking for youth hostels whose price is poorly correlated with those at fancy hotels. What the user needs is a characterization of a portion of the data (which she has identified to the system) along dimensions that are of interest to her.

Let us consider an example task to better appreciate our problem. For variety, we describe a car purchase task rather than a hotel room booking task. For specificity, we write all queries in SQL, even though we expect any real implementation to have a user-friendly interface layer on top the query language.

EXAMPLE 1. *Consider a used car database, which contains a single table $\mathcal{D}$ with n attributes where each tuple represents a car for sale. The table has numerous attributes that describe details of the car, such as* `Price`, `Make`, `Model`, `BodyType`, `Drivetrain`, `Mileage`, `EngineSize`, `NumCylinders`, `Color`, `FuelEconomy`, `Power`, `Year`, *etc.*

Consider a user Mary who is unfamiliar with cars and wants to buy a relatively new SUV car. She has five initial Make *preferences (*Ford, Chevrolet, Toyota, Honda *and* Jeep*), because she has friends who drive these* `Make`*s, but she is open to explore any similar option. She starts her exploration*

*with an initial query: $\mathcal{R}$ = SELECT * FROM $\mathcal{D}$ WHERE Mileage BETWEEN 10K AND 30K AND Transmission = Automatic AND BodyType = SUV. This query leads to a large result set with thousands of tuples. Mary has to specify more constraints to get to a smaller result set that she can explore in depth. She thinks a good place to start may be to reduce the number of* `Make`*s she considers. Let's look at her difficulties in choosing between* `Make`*s.*

Limitation 1. Understanding Attribute Values — The attribute `Make` has more than 50 values. Even to choose among the 5 `Make`s she has initially chosen, she needs to understand what is the main difference between SUVs from any pair of manufacturers, such as Jeep and Chevrolet. Furthermore, Mary knows that her initial set of 5 `Make`s is just a rough starting point, so before she narrows it down further she may also want to understand what other `Make`s are similar, and therefore belong in her consideration set. For example, she may want to know who else makes SUVs very similar to those made by Chevrolet.

Comparison can be of two types: *independent* and *conditional*. An independent comparison would be comparing the general characteristics of Chevrolet vs. Jeep cars. A conditional comparison would be based on the user's already made selections. For example, Mary might want to compare the five `Makes`: Ford, Chevrolet, Toyota, Honda and Jeep, given the following choices: `BodyType` = SUV, $10K \leq$ `Mileage` $\leq 30K$, `Transmission` = Automatic. Conditional comparisons are difficult even for users who are quite familiar with the domain. For example, Mary might know what `Make` she would prefer if there are no constraints on other attributes. However, if there are constraints in other attributes, such as `Price`, `Year`, `Color`, etc., it is hard to find `Make`s that will lead to cars that maximally satisfy her preferences across all the attributes.

With traditional relational database, result sets are presented as sets of tuples. To compare Chevrolet SUVs with Jeep SUVs, Mary has to look at hundreds of instances in each set. This is very difficult to do. Perhaps these tuples could be sorted on some important attributes, such as `Price`, so that corresponding tuples can be compared. But this requires knowing enough of the database to choose important attributes, and even so provides only limited assistance in understanding.

Finding similarities and differences are two complementary aspects of comparison. We looked at the difference case in the preceding paragraph. To find additional similar `Make`s is even more difficult, because now we need to check the hundreds of Chevrolet instances with the thousands of instances from dozens of other manufacturers. We note further Mary has a desired mileage range initially specified. As she explores the data set, she may decide to change this. If she is comparing `Make`s conditioned on her mileage selection, then she has a whole new comparison. The conditional comparisons change with every change in the given query condition.

Limitation 2. Querying Hidden Attributes — Often, there are characteristics of the data item that are important to the user but not explicitly recorded as an attribute in the database. For example, Mary wants to choose a certain car body look, but this field is not encoded anywhere in the database. There may be a way to express her preference as a selection on available attributes (perhaps as a combination of low height, large wheel diameter and four doors). But Mary does not know how to express her desired look in terms of these other attributes.

Even worse, many database interfaces, for the sake of simplicity, may limit the number of queriable attributes. The number of cylinders in the engine may be an attribute recorded in the database, but it is not available to Mary through her forms-based interface for querying the database. It is possible that queriable attributes, such as fuel efficiency, can be used as surrogates to express her preference for a 4 cylinder engine. However, such cross-attribute relationships are completely opaque to Mary, and she is unable to substitute the surrogate for her desired attribute.

In exploring a database, users have two problems: (a) Choosing attributes that will enable them to efficiently and precisely reach to their desired result set, and (b) Choosing attribute values for each chosen attribute. These choices are challenging because there is complex dependency between attribute values within and across attributes, and users also have an unspecified, complicated preference function that spans across multiple attributes. Moreover, their preference function changes on seeing the comparison between available choices.

In short, while exploring a data set, users often make choices in sequence (with some backtracking where needed). They need help to understand the fragment of the database that is currently selected, and they would like to see this fragment of the data set characterized in terms of the choices (of attributes and attribute values) that the user is contemplating next. The alternation of browsing and querying in user interaction with data has been well-documented, where the purpose of browsing is mostly exploration. When data sets become large, unfortunately, browsing is no longer effective because of the very large number of tuples to be viewed. Therefore, this understanding of selected database fragment is best provided as a context-sensitive summary that supports the user's exploration need.

In this paper, we present a novel data summarization technique called the *Conditional Attribute Dependency* (CAD) View, which allows users to systematically explore the conditional dependencies between attribute values both within and across attributes. It thereby lifts the two limitations described in the motivating example above. Our proposed CAD View can be integrated with any structured data presentation system.

Our key contributions are as follows:

- We identify two limitations that users face in exploring databases due to limited data familiarity (Section 1).
- We propose a query model and a data summarization technique called the CAD View that can help users gain familiarity with structured datasets (Section 2).
- We present the algorithms and techniques necessary to create and present the relevant CAD Views (Sections 3 and 4).
- We integrate CAD View with faceted navigation to make the exploratory search process user-friendly. Moreover, this also leads to a novel search interface that can support both exploratory and lookup search. (Section 5).
- We evaluate the CAD View on real data with a detailed user study. We find that users, on average, can perform

Make	Compare Attrs.	IUnit 1	IUnit 2	IUnit 3
Chevrolet	Model Engine Price Drivetrain Year	[Traverse LT] [Equinox LT] [V6] [25K-30K] [20K-25K] [AWD] [2011-2012]	[Suburban 1500 LT, Tahoe LT] [V8] [35K-40K] [40K-45K] [4WD] [2WD] [2011-2012]	[Captiva LS, Equinox LT] [V4] [15K-20K, 20K-25K] [2WD] [2011-2012]
Ford	Model Engine Price Drivetrain Year	[Escape XLT] [Escape Ltd.] [V6, V4] [20K-25K, 15K-20K] [2WD][4WD] [2011-2012] [2010-2011]	[Explorer XLT] [Explorer Ltd.] [V6] [V8] [30K-35K] [25K-30K] [4WD] [2WD] [2011-2012]	[Edge Ltd.] [Edge SEL] [V6] [25K-30K] [AWD, 2WD] [2011-2012] [2010-2011]
Honda	...	...	...	...
Toyota	...	...	...	...
Jeep	Model Engine Price Drivetrain Year	[Wrangler Unlimited] [V6] [V8] [25K-30K] [30K-35K] [4WD] [2011-2012] [2010-2011]	[Compass Sport, Patriot Sport] [V4] [15K-20K] [4WD] [2WD] [2011-2012]	[Liberty Sport] [V6] [15K-20K] [4WD] [2WD] [2011-2012] [2010-2011]

Table 1: This table shows a sample Conditional Attribute Dependency (CAD) View for comparing five different car manufacturers. The first column `Make` is the Pivot Attribute. The second column `Compare Attributes` shows the top-5 attributes that are most informative for comparing the five `Makes`. The last three columns shows the top-3 IUnits for each `Make`. The user has selected `BodyType` = SUV, $10K \leq$ `Mileage` $\leq 30K$, `Transmission` = Automatic. Each IUnit is a cluster label that summarizes a group of similar SUVs.

tasks that require data understanding with 4-5 times greater efficiency and accuracy using our CAD View as compared to faceted interface (Section 6).

Finally, we conclude with Section 8 after a discussion of related work in Section 7.

2. SOLUTION ARCHITECTURE

In this section we describe our solution to the exploratory search problem in complex databases — the *Conditional Attribute Dependency* (CAD) View. We also identify algorithmic challenges that must be solved for the CAD View to fulfill its goals.

2.1 The CAD View

The CAD View is a novel data summarization technique that shows the conditional relationship between values in a given attribute with values in other attributes. It is best introduced by example. A more formal treatment follows in the next subsection.

2.1.1 *Overview*

Table 1 shows a sample CAD View, obtained from a real dataset, for the example query discussed in Section 1. Mary's goal was to explore automatic transmission SUV cars that have `Mileage` between 10K-30K from five different `Makes`. The CAD View has several important components:

1. The Pivot Attribute organizes the information that is shown in the CAD View. A user explicitly chooses one of the attributes as *Pivot Attribute* f_p and requests the system to create a CAD View that facilitates comparison among attribute values selected from the Pivot Attribute by showing their relationship with values across other attributes. In Table 1, Mary has chosen Make as the Pivot Attribute.

2. Compare Attributes are data attributes that interact with the Pivot Attribute in "interesting" ways. All the values in the Pivot Attribute are compared using the same set of Compare Attributes. These attributes can be automatically determined based on the result set and the Pivot Attribute or explictly provided by the user. For example, one can use correlation to quantify interesting interaction. In Table 1, the system has given five Compare Attributes: Model, Engine, Price, Drivetrain, and Year.

3. An IUnit (Interaction Unit) is an "interesting" group of values for the Compare Attributes. In Table 1, each IUnit is described using the five Compare Attributes mentioned above. Each IUnit is chosen to be relevant to a Pivot Attribute value: Chevrolet, Ford, Honda, etc. The top-left IUnit in this table (containing Traverse LT and Equinox LT) identifies a set of midsized Chevrolet SUVs: they share an engine size and a drivetrain, and have similar prices. One can think of an IUnit as a cluster of database values with two special differences: it is a cluster on a partition of the database determined by each Pivot Attribute value, and the cluster is labeled using the chosen Compare Attribute labels and Compare Attribute values.

The Overall CAD View is a tabular combination of the above three components. It displays one row for each value of the user-selected Pivot Attribute. In the second column the system shows an ordered list of Compare Attributes, one for each row of the table. The rest of the table shows each row's top IUnits, sorted left-to-right in descending order of relevance to the row's Pivot Attribute value. If an IUnit cluster can be represented equally well by multiple values in a single Compare Attribute, then an IUnit will show multiple attribute values in square brackets (e.g. Traverse LT and Equinox LT).

Note that there are competing ways to rank IUnits from left-to-right within each row. They can be ranked left to right in order of their salience for the row's Pivot Attribute value. Or we could try to ensure that all of the IUnits in a single column can be compared across all Pivot Attribute

values so that, *e.g.*, the IUnit 1 for Chevrolet is similar to IUnit 1 for Ford (and thereby addressing Limitation 1).

However, not all Pivot Attribute values may share comparable IUnits, forcing our system into an impossible tradeoff between IUnit quality and columnar IUnit "comparability." Thus, we chose to rank IUnits strictly by their relevance to the row's Pivot Attribute value. We use other means to satisfy the comparability goal as described below in Section 2.1.3.

2.1.2 Query Model

We use the following extension of SQL to express an exploratory search query:

```
CREATE CADVIEW cadview_name AS
  SET pivot =  pivot_attr
  SELECT attr1, attr2,...,attrN
    FROM table1, table2...
    [WHERE Clause]
    [LIMIT COLUMNS M] [IUNITS K]
    [ORDER BY attr_name, attr_name ASC|DESC]
```

In the above expression, the list of attributes shown in the SELECT clause are the attributes that the user has explictly selected as Compare Attributes. The LIMIT COLUMNS clause is used to limit the number of Compare Attributes. The CAD View will have total of M columns as Compare Attributes, in which N are explicitly provided by the user and the remaining (M-N) are automatically selected based on the query result and the Pivot Attribute. The number of IUnits per row K is determined using the keyword IUNITS. The ORDER BY keyword can be used to sort the IUnits by one or more columns.

```
CREATE CADVIEW CompareMakes AS
  SET pivot =  Make
  SELECT Price
    FROM UsedCars
    WHERE Mileage BETWEEN 10K AND 30K AND
      Transmission = Automatic AND BodyType = SUV AND
      (Make = Jeep OR Make = Toyota OR Make = Honda OR
       Make = Ford OR Make = Chevrolet)
    LIMIT COLUMNS 5 IUNITS 3
```

For example, Mary's query can be expressed as above. The `Price` attribute has been explicitly selected as a Compare Attribute, and the remaining four attributes (`Model`, `Engine`, `Drivetrain` and `Year`) are automatically determined.

2.1.3 Finding Similar Information

If there are $\mathcal{V}$ values in the Pivot Attribute and the user has requested k IUnits per attribute value, then the CAD View will have $k|\mathcal{V}|$ IUnits. As discussed in Section 1, one of the primary goal of exploratory search is comparison, which includes finding similarities and differences. To facilitate comparison, we support the following two search operations within the CAD View: (i) Finding similar top ranked IUnits, and (ii) Finding similar attribute values within the Pivot Attribute.

For example, if a user likes a particular IUnit from one of the selected Pivot Attribute values (e.g., Chevrolet), then the user may want to efficiently locate similar top-ranked IUnits that belong to other Pivot Attribute values. Similarly, if the user likes multiple IUnits of a particular Pivot Attribute value, then the user might be interested to find out other Pivot Attribute values that have similar IUnits.

Let's say Mary likes IUnit 3 of Chevrolet. She can create a new CAD View where all the IUnits that are similar to this IUnit gets highlighted by using the following query:

```
HIGHLIGHT SIMILAR IUNITS
  IN CompareMakes
  WHERE SIMILARITY(Chevrolet, 3) > 3.5
```

In the similarity function the user gives the Pivot Attribute value and the IUnit ID. The above query will highlight all the IUnits (e.g., IUnit 1 of Ford, IUnit 2 of Jeep) in the CAD View CompareMakes with similarity score greater than 3.5. As discussed later, for five Compare Attributes the max similarity score can be 5.0.

Similarly, to find `Make`s that are similar to Chevrolet, one can reorder the rows of the CAD View such that the Pivot Attribute values are ordered in terms of decreasing similarity with respect to Chevrolet. The similarity between two Pivot Attribute values can be measured by measuring the similarity between their IUnits. This query can be expressed as follows:

```
REORDER ROWS
  IN CompareMakes
  ORDER BY SIMILARITY(Chevrolet) DESC
```

2.1.4 Design Goals

We can now examine the extent to which the CAD View addresses the limitations described in the motivating example above:

Limitation 1. Understanding Attribute Values — With the traditional tuplewise presentation of result set, it was difficult for Mary to find the `Make`s that are similar to Chevrolet, or see the difference between Chevrolet and Jeep. However, using the CAD View it is easy to see that IUnits of Chevrolet and Ford are quite similar, and thus one can infer that both Chevrolet and Ford offer SUVs at roughly similar capacities and price points. One can also see that SUVs from Chevrolet and Jeep are quite different, and they primarily differ in `Price` and `Drivetrain`. Moreover, the CAD View can show conditional comparisons. Since Mary had selected `Mileage` between 10K and 30K, the CAD View shows her comparison between SUVs in `Year` range 2011-2012.

Limitation 2. Querying Hidden Attributes — Also recall that Mary was unable to choose cars with V4 engines, because the interface did not expose Engine type as an option in the query panel even though the information was contained in the database (*i.e.*, Engine was a non-queriable attribute). Moreover, she was not familiar enough with the database to indirectly find V4 engines by selecting values in the *queriable* attributes. In contrast, the CAD View identifies V4 engines as a characteristic of specific IUnits for each body style. Mary can select the desired tuples using the corresponding queriable attributes.

2.2 Problem Definition

2.2.1 Assumptions

The CAD View is a tabular structure whose size must be small enough for the summary information to be absorbed effectively by the user. For example, the width must be small enough not to require horizontal scrolling when displayed on the user's screen. We reflect this constraint on the width by limiting the number of IUnits we can show for each attribute

value. Let this number be k. We assume that k is given to us, either by the user explicitly, or through the system gaining knowledge of the user's set up.

The length of the table must also be constrained for the same reason. There are two variables that control table length. The first is the number of distinct values for the pivot attribute. By default, we will show all of them. If the user is focused on only specific values, these can be listed explicitly in the CAD View specification. Mary has chosen 5 specific Makes in the example above. The second variable affecting table length is the number of Compare Attributes in each row. We assume that this number c is given to us, just as k is. Furthermore, if the user is interested in specific attributes, she can insist that these be included in the Compare Attributes that the system selects.

For categorical attributes or attributes with small discrete numerical domain, the attribute values are directly obtained from the domain. Where the number of values is very large, such as for most numerical domains, ranges of values are binned together to create a small number of discrete attribute values. Such attribute value cardinality reduction is necessary for effective summarization. However, this cardinality does not itself play a role in the CAD View generation algorithm. Therefore, we mention it here as a pre-processing step, but do not go into details of exactly how this binning is done. We suggest following the well-developed techniques in histogram construction[17] for this purpose.

In this section we describe the problems that needs to be solved to create the CAD View. Our goals are (i) to populate this structure effectively, making the most of limited screen real-estate available, and (ii) to arrange and present the information populated in this structure to maximize its value to the user.

For the first goal, we have to find the best (i.e., most informative) Compare Attributes, the best IUnit clusters, and (for each IUnit) the best value labels to describe the IUnit's data.

The CAD View structure already lays out IUnits in rows, one per attribute value for the Pivot Attribute. For the second goal, the system must further decide how to order IUnits within each row, how to indicate similarity between IUnits in different rows, and how to indicate similarities and differences between rows as a whole.

2.2.2 Creating the CAD View

The CAD View is created for a given result data set $\mathcal{R}$ and a Pivot Attribute. Populating the CAD View entails one main task: obtaining the k IUnits of interest for each value of the pivot attribute. This task can be written formally as:

Problem 1 (Generate IUnits): *Given a result set $\mathcal{R}$, a Pivot Attribute f_p, a set of attribute values $\mathcal{V}$ selected from f_p, and a threshold value k, find for each attribute value $v \in \mathcal{V}$ a list of k IUnits S^v, where $S^v = \{s_1^v, s_2^v, ..., s_k^v\}$ and s_j^v is the jth IUnit for attribute value v.*

The first task could be accomplished as finding k clusters with our favorite clustering algorithm. However, we observe that our goal is to explain the main structure of this fragment of the data set to the user. Therefore, there are two important ways in which we deviate from the basic problem statement above. The first is that we restrict the clustering to be on the basis of only the attributes selected as *Compare Attributes*. These are the attributes that will be displayed in the CAD View. In other words, these are the attributes that will be used to label each cluster (IUnit). Therefore, it is the values of these attributes that we wish to have clustered together in each IUnit rather than some other attributes not shown to the user. The second point we note is that we are under no obligation to cover all points in the data set with the clusters produced. We do not want outliers to distort the clusters. To this end, we choose to solve the clustering problem with a larger number l, and then choose the top-k IUnits from among these l clusters. l can be chosen by iterating through all plausible l values and evaluating the quality of the resulting CAD View for each. Or it could be obtained as a system tuning parameter, such as $l = 1.5k$.

We can then restate the CAD View generation problem as the following sequence of sub-problems:

Problem 1.1 (Compare Attributes): *Given a result set $\mathcal{R}$, a Pivot Attribute f_p, and set of attribute values $\mathcal{V}$ selected in f_p, find a subset of Compare Attributes $\mathcal{I}$ s.t. $\mathcal{I}$ generate the most contrast among values in $\mathcal{V}$.*

Choosing Compare Attributes is a feature selection problem [12, 22] with a specialized way of evaluating the quality of a feature: good features (that is, Compare Attributes) yield sharply contrasting IUnits across the different Pivot Attribute values. One can discriminate among Compare Attributes as follows: Given a multi-class problem, a feature X is preferred to another feature Y if X induces a greater contrast between the multi-class conditional probabilities than Y. X and Y are indistinguishable if they induce the same amount of contrast.

Problem 1.2 (Generate Candidate IUnits): *Given a result set $\mathcal{R}$, a Pivot Attribute f_p, a set of attribute values $\mathcal{V}$ selected from f_p, a set of Compare Attributes $\mathcal{I}$, and a threshold value l, find for each attribute value $v \in \mathcal{V}$ a list of l candidate IUnits S^v, where $S^v = \{s_1^v, s_2^v, ..., s_k^v\}$ and s_j^v is the jth candidate IUnit for attribute value v.*

Problem 1.2 is now stated as a clustering problem, with each resulting cluster being a candidate IUnit. We finally need to choose k IUnits from among these l candidates.

Problem 2 (Top-k IUnits): *Given a list of IUnits S^v for attribute value v, and a preference function P, find the top-k IUnits T^v in S^v according to preference P, where $T^v = \{t_1^v, t_2^v, ..., t_k^v\}$ and $T^v \subseteq S^v$.*

The IUnits could be ranked based on a function that is rooted in the clustering algorithm; for example, we could prefer "tight" clusters by ranking them in ascending order of minimum pairwise similarity. However, we can pursue some application-specific goals by ranking IUnit clusters in a manner that is distinct from the IUnit creation mechanism. For example, our car navigation interface might, by default, rank clusters in ascending order of cluster price. In contrast, the fleet manager for a taxi company might have a different preference function that ranks IUnits in descending order of car mileage. Therefore, we have defined this ranking in terms of a specific preference function. If no function is specified by the user, we can use a simple system default, such as cluster size.

2.2.3 Finding Similar Information

The two search operations within the CAD View can be stated as following two problems:

Problem 3 (Similar IUnits): *Given two attribute values*

x and y from the Pivot Attribute, and an IUnit t_i^x from T^x, find all IUnits t_j^y s.t. $t_j^y \in T^y$ and $\mathsf{sim}(t_i^x, t_j^y) \geq \tau$.

We can use any similarity function for this purpose, and any user or system specified threshold τ. We describe the specifics of the similarity function in Section 4.1.

Problem 4 (Similar Attribute Value): *Given two attribute values x and y and their top-k list of IUnits T^x and T^y, find the similarity between x and y by measuring the similarity of their top-k IUnits.*

If a user shows preference for a particular attribute value, it implies that the user has liked most of the top-k IUnits that has been shown for that attribute value. The user would be interested to see other attribute values that have similar IUnits both in terms of content and rank. We describe the specifics in Section 4.2.

3. CREATING THE CAD VIEW

In this section, we describe how we create and sort IUnits (problems 1 and 2 above) for the CAD View.

3.1 Generating Candidate IUnits

Generating uniformly labeled IUnits consists of two steps: finding good Compare Attributes $\mathcal{I}$ that can show contrast in Pivot Attribute values $\mathcal{V}$; and then generating l IUnits for each value $v \in \mathcal{V}$.

3.1.1 Finding Compare Attributes

The problem of finding Compare Attributes is similar to feature selection in a multi-class classification problem. To provide efficient user interaction and understanding, we use a feature selection algorithm that is computationally efficient and returns all the relevant features.

To determine the number of Compare Attributes we consider two factors: the available screen space and the relevance score of each informative facet. The user's available screen space determines the maximum number c of Compare Attributes that can be shown for any Pivot Attribute. However, all Pivot Attribute may not have c informative facets that have relevance greater than a required minimum threshold relevance score. A relevant Compare Attribute always provides additional useful information. However, if a Compare Attribute is not informative about the Pivot Attribute, including it will lower the quality of generated IUnits and waste valuable screen space.

We use the ChiSquare feature selection algorithm [23]. ChiSquare evaluates the worth of an attribute by computing the value of the chi-squared statistic with respect to the class. For ChiSquare test one can determine the threshold relevance using p-values, such as significance level equal to 0.01, 0.05, or 0.10. Even with this simple technique, ranking Compare Attributes in order of decreasing relevance yields a few interesting observations that a typical user might not know. For example, it might seem that `Mileage` should be the best Compare Attribute when distinguishing among different `Year` values: older cars will naturally accrue more miles. However, it turns out that Model is better, as specific car models (Suburban 1500 LT, not simply Suburban) are released frequently, and a specific model is prominent in the database for only a short period of time.

3.1.2 Finding Important Attribute Interactions

To create IUnits for a Pivot Attribute value $v \in \mathcal{V}$, we take all tuples from the result set $\mathcal{R}$ that contain the given value v, and allocate those tuples to l clusters. We derive an IUnit from each of these l clusters. We cluster the tuples using only the above-chosen Compare Attributes.

Since the CAD View is a user-facing application, we want to create it within interactive time limits, well under 1 sec. There are various factors that can slow down a clustering algorithm: (i) clustering a dataset with large numbers of tuples or dimensions, (ii) trying to infer the ideal number of clusters using the clustering algorithm, and (iii) clustering with large numbers of cluster centers. Since there are standard existing techniques to address each of these factors, we defer their discussion to experimental evaluation (Section 6.3).

The quality of IUnits depends on the quality of the clustering algorithm. Since both efficiency and quality are major concerns of our system, we use standard k-means algorithm. Our main contribution in the clustering step is the dynamic variation of system parameters to achieve real-time performance, as discussed later in Section 6.3.

Our key contribution in creating the IUnits is the post-clustering step of cluster labeling, which is often ignored in clustering research. Although clustering is a very nice data-categorization technique, it is very hard for most users to understand the large amount of information that is contained in each cluster, or be able to compare multiple clusters.

There are existing systems to visually explore clusters of structured data [5, 21, 29]. Some of these systems are not easy to explore when the data is high-dimensional or categorical. For normal end-users, the commonly used cluster labeling technique is to show the centroid of each cluster, which is useful when all clusters are spherical. For complex shaped clusters, it is considered more informative to show multiple tuples that can show the whole cluster boundary [5]. It is very hard to understand a high-dimensional cluster by seeing just one centroid or some boundary points. When a user sees a high-dimensional representative tuple, it is not easy to infer the dimensions that are most significant. We need to label the clusters in such a way that we can convey large amount of information in a summarized manner and also emphasize the important information.

The way we label the clusters has many benefits. We label all IUnits uniformly and use ranking at all levels. We rank the Compare Attributes to highlight the attributes that are most significant. Similarly, in each IUnit we rank the Compare Attribute values and show only the most important representatives. Instead of showing few representative tuples from each cluster, we try to summarize statistical distribution of each Compare Attribute. To label both categorical and numerical attributes in uniform manner, we discretize the numerical attributes. We rank attribute values based on frequency count and then group multiple values if they have similar frequency count. We use two thresholds — max display count and statistical difference between frequency counts — to determine the representative Compare Attribute values for each cluster.

3.2 Top-k IUnits

Without an explicit user preference function, we choose a preference function that depends on the size of the IUnit's underlying cluster, as well as overall result diversity. IUnits that represent large clusters are desirable because they

summarize attribute interactions for larger number of tuples. Moreover, they may give more reliable insight than smaller outlier-prone clusters. However, when we select the top-k IUnits based purely on the cluster size, many are quite similar and appear redundant to the user.

Thus we use the generic top-k algorithm proposed in [25] to compute diversified top-k IUnits. It requires the following measures: preference score of each IUnit s_i^x, denoted as $\mathsf{score}(s_i^x)$; similarity between two IUnits s_i^x and s_j^y, denoted as $\mathsf{sim}(s_i^x, s_j^y)$; and a user defined threshold similarity value τ. Two IUnits s_i^x and s_j^y are considered similar, denoted as $s_i^x \approx s_j^y$, if $\mathsf{sim}(s_i^x, s_j^y) \geq \tau$.

Diversified Top-k IUnits: *Given a list of IUnits $S^v = \{s_1^v, s_2^v, ...\}$ for a attribute value v, and an integer k, the diversified top-k IUnits for v, denoted as $T^v = \{t_1^v, t_2^v, ...\}$, is the list of IUnits that satisfy the following conditions:*
1) $T^v \subseteq S^v$ and $|T^v| \leq k$
2) For any two different IUnits s_m^v and s_n^v, if $s_m^v \approx s_n^v$ then $\{i_m^v, i_n^v\} \not\subseteq T^v$
3) $\Sigma_{t_i^v \in T^v}$ $\mathsf{score}(t_i^v)$ is maximized.

To create the CAD View, we need to compute the diversified top-k IUnits for each attribute value v. The diversified top-k problem can be reduced to the NP-Hard maximum independent set problem on graphs [25]. Greedy solutions often lead to good approximate results in many NP-Hard problems, but for this problem a greedy algorithm can lead to arbitrarily bad solutions, with no bounded constant factor solution [25]. Because in our problem the size $|S^v|$ is generally not large, Qin, *et al.'s* basic `div-astar` algorithm works well.

4. FINDING SIMILAR INFORMATION

In this section, we decribe how to find similar components in the CAD View. These are solutions to Problems 3 and 4.

4.1 Finding Similar IUnits

If a user likes one of the IUnits, say IUnit t_i^x from T^x, the user can find all the IUnits t in the CAD View s.t. $t_i^x \approx t$ (in other words, $\mathsf{sim}(t_i^x, t) \geq tau$). This approach allows us to address the IUnit sorting problem mentioned in Section 2.1.1; we can now sort IUnits from left-to-right by order of salience to the row's Pivot Attribute value, while still allowing the user to compare similar IUnits.

Computing similarity between IUnits is equivalent to computing similarity between clusters. For a numerical dataset, one can compute cluster distance by measuring the distance (such as Euclidean distance) between cluster centroids. For a categorical dataset, one can use any distance measure that is used in existing categorical clustering algorithms to compute cluster distance [11]. However, things become more complicated when we have a mixed dataset, having both numerical and categorical attributes. The distance measure that is used in categorical datasets is quite different compared to those used in numerical datasets. To compute similar IUnits, we propose a new distance measure that can treat both numerical and categorical attributes in a uniform manner. We use discretization to convert numerical attributes into categorical attributes. Then we use a modified form of *cosine-similarity* to compute IUnit similarity.

Let t_i^x and t_j^y be two top-k IUnits for selected attribute values x and y, and $\mathcal{I}$ be their set of Compare Attributes. We measure the similarity of t_i^x and t_j^y by summing their cosine

Algorithm 1 IUnit Pair Similarity

Input: t_i^x: IUnit 1
t_j^y: IUnit 2
$\mathcal{I}$: set of informative dimensions
Output: s: similarity between the two IUnits
Method:

```
1: s ← 0
2: for all d ∈ I do
3:     s ← s + cosine-similarity(t_i^x.d, t_j^y.d)
4: end for
5: return s
```

similarities along each dimension d s.t. $d \in \mathcal{I}$. We use the frequency count of each attribute value in the corresponding cluster as the attribute value's term frequency. Since the range of *cosine-similarity* function is $[0, 1]$, the range of the above similarity function is $[0, |\mathcal{I}|]$. Based on the specific data domain, one can choose the IUnit similarity threshold value τ as some $\alpha.|\mathcal{I}|$, where $\alpha \in (0, 1)$.

Algorithm 2 Attribute-value Pair Similarity

Input: $T^x = \{t_1^x, t_2^x, ..., t_k^x\}$ top-k IUnits for attribute value x
$T^y = \{t_1^y, t_2^y, ..., t_k^y\}$ top-k IUnits for attribute value y
Output: d: distance between T^x and T^y
Method:

```
1: d ← 0
2: for all t_i^x ∈ T^x do
3:     if ∃t ∈ T^y s.t. t ≈ t_i^x then
4:         index ← j s.t. t_i^x ≈ t_j^y and argmin_j |j − i|
5:     else
6:         index ← |T^y| + 1
7:     end if
8:     d ← d + |i − index|
9: end for
10: for all t_j^y ∈ T^y do
11:     if ∃t ∈ T^x s.t. t ≈ t_j^y then
12:         index ← i s.t. t_i^x ≈ t_j^y and argmin_i |j − i|
13:     else
14:         index ← |T^x| + 1
15:     end if
16:     d ← d + |j − index|
17: end for
18: return d
```

4.2 Finding Similar Attribute Values

If a user has preference for a Pivot Attribute value, the user can create a CAD View where the first row contains IUnits for the preferred value, and the remaining Pivot Attribute values are shown in decreasing order of similarity to the preferred value. Two attribute values are considered similar if their top-k IUnits lists are similar. Two ranked IUnit lists T^x and T^y should be similar if they have similar IUnits, and similar IUnits have similar rank.

To the best of our knowledge, there is no existing distance metric to compute similarity between two ranked lists having a disjoint set of items. In Algorithm 2, we propose a distance measure that can compute distance between two given ranked lists by taking into consideration the similarity between their items both in terms of information content and rank.

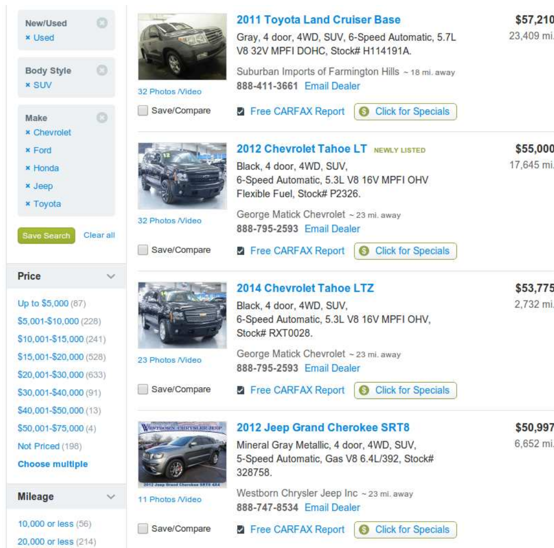

Figure 1: This screen capture from `cars.com` represents an example of a faceted navigation interface.

In lines 2-9, we compare how IUnits in T^x compare to IUnits in T^y. In line 3, we check whether the list T^y has some IUnit which is similar to IUnit t_i^x from T^x. If there is no similar IUnit (line 6), we assume that t_i^x is similar to the IUnit that has highest rank amongst non-selected IUnits (i.e., $S^y \setminus T^y$), and thus has rank $|T^y| + 1$. In lines 3-4, if there are multiple IUnits in T^y that are similar to t_i^x, then we take the IUnit whose rank is closest to rank of t_i^x in T^x, which is i. In line 8, we sum the rank differences for all IUnits in T^x. Lines 10-16 show the same steps for list T^y.

5. FACETED SEARCH WITH CAD VIEW

The CAD View defined above can be used with any relational database, independent of any front ends used. In fact, we have even suggested small SQL extensions to capture this concept. Nevertheless, we recognize that most end-users are unlikely to be SQL programmers, and are likely to be accessing relational data through some user-friendly interface. In this section, we consider one such popular interface, and describe how we have integrated CAD View with it.

Shoppers in e-commerce applications are a major target for our work: they are often exploring unfamiliar web sites before they actually buy. Since most e-commerce web sites use faceted navigation, that is the interface that we chose to integrate CAD View into. Figure 1 is a screenshot of a typical faceted interface for browsing a database of cars. In this section, we describe a novel two-phased faceted interface, called *TPFacet*, which integrates CAD View with faceted browsing.

A basic faceted interface has two main component panels: a query panel and a results panel. The latter typically occupies the majority of the screen real estate and shows the set of currently selected items. The former is usually on the left side, and offers both user interface controls as well as a summary digest of the current query and result set. This summary digest typically comprises all the attribute values (attribute values) that appear in the selected items, grouped by their corresponding attribute (attribute). The tuple count for each attribute value may also be included.

To fit the CAD View within users' limited screen space, we propose a slightly changed interaction model for faceted navigation: at any one time, the interface will display either the results panel *or* the CAD View. The user explicitly toggles between them, though it is easy to imagine a system that intelligently chooses a default view, based on the size of query results. We imagine the user will interact with the system in two distinct phases: the *query revision* phase focuses on the CAD View, while the *result set* phase focuses on the results panel, with the user exploring individual items of interest in the result set.

Faceted navigation is an interaction based search system. We need to modify the faceted search interface so that users can create the CAD View or find similar components within the CAD View using interactive search techniques. We made the following three modifications: (i) Make each queriable attribute selectable (using html radio buttons) so that users can select them as Pivot Attribute, (ii) When users click on an IUnit in the CAD View we highlight all the other similar IUnits, and (iii) When users click a Pivot Attribute value in the CAD View we reorder all the rows in the CAD View in decreasing order of similarity w.r.t. the clicked attribute value. We call the faceted interface with these changes as TPFacet system.

6. EVALUATION

The goal of the CAD View is to facilitate exploratory search in complex datasets. As such, the primary evaluation of the CAD View is by means of a user study. In particular, we compare the use of the CAD View with a standard faceted interface for three exploratory search tasks. As a baseline for comparison, we use Apache Solr [2], which is a popular open source enterprise search platform. Apache Solr has support for faceted navigation and is used by many e-commerce sites. Apache Solr has many configuration settings. We chose a setting that is closest to the CAD View query model. We discuss the user study in depth in Section 6.2.

A secondary question is one of performance. Since the summaries shown in the CAD View are quite complex, we have to make sure that they can be computed in reasonable (interactive) time for the data set complexities and sizes that we expect. We discuss this issue in Section 6.3.

6.1 Implementation and Environment

We integrated the CAD View with Apache Solr to design the TPFacet system (see Section 5). We input the users' query from faceted interface, compute the CAD View and all similarity scores in the backend server, and return the resulting CAD View and similarity information using HTML and Javascript. To do feature selection and clustering, we use ChiSquare and SimpleKMeans algorithm respectively. Both algorithms are available in Weka [13].

We used two real datasets—YahooUsedCar and Mushroom [9]— to do the evaluation. We scraped Yahoo's used car site [1] to create a table comprising 40,000 tuples with 11 attributes. The Mushroom dataset has 8124 tuples with 23 attributes. These numbers are at the lower end of what one sees in a typical e-commerce dataset. The CAD View will become more valuable in datasets that have more number of attributes or tuples. The Mushroom dataset is very popular in machine learning. It is simple to understand for a non-expert, since it describes familiar properties, such as color and smell, but has data that most of us (and all of our

users) have no knowledge of, forcing us to learn patterns by examining the data set afresh without reliance on previous knowledge.

6.2 User Study

We devised a diverse set of carefully specified information exploration tasks, described in the subsections that follow, each of which tests (some aspects of) the users' understanding of the database. These tasks roughly correspond to the two motivating limitations discussed in Section 1. The first two tasks correspond to Limitation 1, where we evaluate users' ability to perform comparisons in the form of finding differences and similarities respectively. The third task corresponds to Limitation 2, where we test users' ability to query non-queriable attributes using available queriable attributes. We used the MUSHROOM data set, which was unfamiliar to all our users.

We compare TPFacet and Solr in terms of their usability in users' task completion time and quality of response to given tasks. For all the tasks we report the results using statistical analysis.

We performed our user study using eight graduate students from our university. As we will see in the following subsections, statistical analysis show that the conclusions we draw from these eight users are statistically significant.

We gave all the users a demo explaining all features of the TPFacet and the steps to do the tasks using both the interfaces. We allowed the users to do the tasks remotely to minimize effect of any environmental factors. We created 3 matched pairs of tasks, one pair for each type described below. We divided the eight users into two equal groups. We indicate each user by their user id $U1$-$U8$. Users with id $U1$-$U4$ were assigned to group 1 and $U5$-$U8$ to group 2. For a task pair (A, B) we asked one of the groups to do task A using TPFacet and task B using Solr. We reversed the task assignment for the other group. In other words, if a task was done by group 1 users using Solr, then the same task was done by group 2 users using TPFacet, and vice versa.

For all the three tasks we have performed linear mixed model statistical analysis [28]. We use Display type as fixed effect and User ID as random effect. Computing p-values for mixed models aren't as straightforward as they are for linear models. The most popular way to obtain p-value is to use the *Likelihood Ratio* test as a means to attain p-values. The logic of the likelihood ratio test is to compare the likelihood of two models with each other. First, the model without the factor that one is interested in (the null model) and then the model with the factor that one is interested in. By comparing these two models, one can determine whether the factor one is considering is significant or not. We use ANOVA to compare the two models.

6.2.1 Simple Classifier

This task illustrates the benefits of the CAD View in finding differences between attribute values. We asked users to build a simple classifier. Classification is an important machine learning problem where given a training data with multiple class labels, one builds a classification model by which one can find the set of classes (categories) a new test observation belongs. In this task, we build a classifier for binary class data. We assume a simple classification model that consists of selecting at most two attribute values that maximizes the number of tuples retrieved from a given target class, and minimizes the number of tuples from the other class. Although problems like classification are rarely done manually for large datasets, human ability in this task demonstrates an understanding of crucial database themes. We evaluate the goodness of the classifier using standard F1 accuracy score. A sample task was to build a classifier for target class `Bruises` = `true`, where the given classes were `Bruises` = {`true`, `false`}.

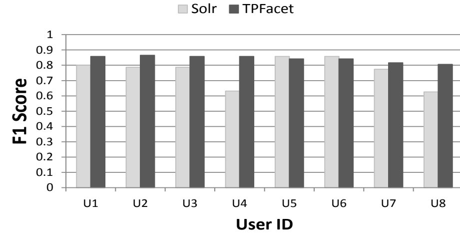

Figure 2: Simple Classifier

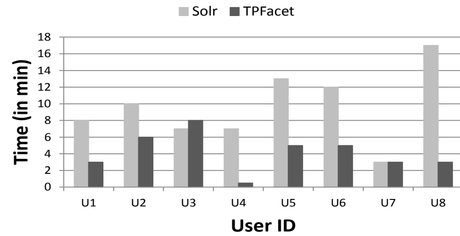

Figure 3: Simple Classifier

Figure 2 shows the F1 scores for the classifiers that users got for this task. Statistical analysis shows that TPFacet affects the quality of classfier by ($\chi^2(1) = 5.572,\ p = 0.018$), increasing the F1 score by about 0.078 ± 0.0285. Moreover, the variation in F1 score is much lower when users use the TPFacet system as compared to Solr because the exploration using TPFacet is more methodical. In Figure 3, we show the time taken by the users to build the classifiers. Statistical analysis shows that TPFacet affects the time taken by ($\chi^2(1) = 8.54,\ p = 0.003$), lowering it by about 5.44 ± 1.56 minutes.

6.2.2 Most Similar Facet Value Pair

This task illustrates the benefits of the CAD View in finding similar (or equivalent) attribute values. In this task, we gave users a list of four attribute values from an attribute and asked them to find the two most similar attribute values. For example, given attribute = `GillColor` and attribute values = {`buff`, `white`, `brown`, `green`}, find the two most similar gill colors.

In the traditional faceted interface, users can Compare Attribute values by comparing their summary digest. We gave users a cosine-similarity based distance metric to compare the summary digests. We asked users to select each of the given attribute values, one at a time, and compare their summary digest. In the CAD View, we didn't show the computed similarity scores, but allowed users to use interactive effects to find similar IUnits and attribute values.

Figure 4 shows users response quality for this task. Since there are four attribute values, there are 6 possible attribute value pairs. We computed the defined similarity score for each pair and ranked them from 1 to 6, with the most similar pair being ranked as 1. Since computing exact similarity score is very hard for humans, we purposely chose attributes

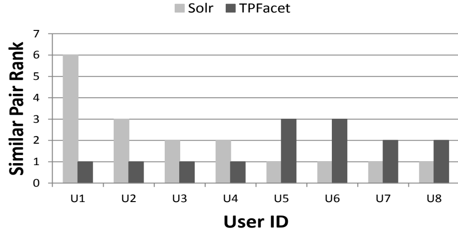

Figure 4: Most Similar Attribute Value Pair

and attribute values that would make the task humanly feasible in Solr. The similarity between gill colors brown and white was so high as compared to other choices that all the eight users got correct answer for this task. Group 1 users ($U1$-$U4$) did this task using TPFacet and group 2 users using Solr. However, the other similarity task was slightly harder. For the other task, users $U7$ and $U8$ got the most similar attribute value pair according to attribute value similarity we defined in Section 4, but according to the metric defined in this task, they turned out to be second most similar pair. Statistical analysis shows that there is no difference in users response quality by using the two types of interface.

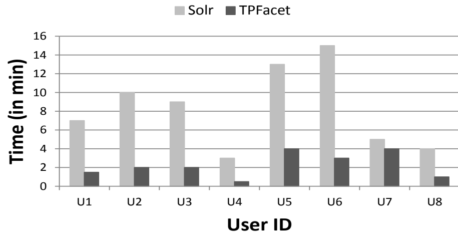

Figure 5: Most Similar Attribute Value Pair

Figure 5 shows the time users took to finish this task. Statistical analysis shows that TPFacet affects the time taken by ($\chi^2(1) = 12.04,\ p = 0.0005$), lowering it by about 6.00 ± 1.23 minutes. All users, except user $U7$, finished the task around four times faster using TPFacet as compared to Solr. Since the users were doing this task for the first time, some of them were trying to manually compare the IUnits. Users could have got the desired answer for this task much faster by just using the interactive effects, as seen in case of users $U4, U8$ and $U1$.

6.2.3 *Alternative Search Condition*

This task illustrates the benefits of the CAD View in querying non-queriable attributes using queriable attribute values. In this task, we gave users a set of selection conditions that lead to some result set $\mathcal{R}$. We asked users to find another set of selection conditions that would lead to same result set $\mathcal{R}$, but not using any of the already given selection conditions. One can see the given selection conditions as selection conditions on non-queriable attributes that the users cannot query. Only an informed user can precisely access the desired result set using an alternate option. A sample task was to find an alternative selection condition using at most two attribute values that would lead to the same result as selecting: StalkShape = enlarged and SporePrintColor = chocolate.

To evaluate users response quality, we checked the similarity between the query result obtained from the given selection condition and the users alternate selection condition. To measure similarity between the two results, we measured the similarity between their faceted summary digest.

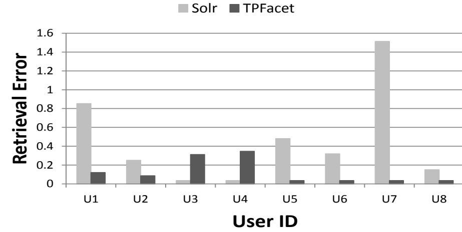

Figure 6: Alternative Search Condition

Figure 6 shows users response quality for this task. Statistical analysis shows that TPFacet affects the users alternative search condition by ($\chi^2(1) = 3.28,\ p = 0.07$), lowering the retrieval error by about 0.329 ± 0.172. Using TPFacet most users were able to do the task with five times lower retrieval error. In this task pair, the task that group 1 users did using Solr, turned out to be quite easier compared to the one they had to do using TPFacet. We can see this difference by seeing that users $U3$ - $U8$ have very low and similar error for this task. For the easier task, just one attribute value was sufficient to get to the desired result set. All the users in group 2 had come up with slightly variant solutions, but exactly the same retrieval error (48 missing tuples out of 1344). Since TPFacet allows users to do this type of task in more methodical approach compared to Solr, we find much lower variation in users response quality. For the slighly harder task, we see slight variation in retrieval error among group 1 users who did this task using TPFacet, but the error variation is much higher for group 2 users who did it using Solr. Group 2 users, such as $U5, U6$ and $U8$, who had much lower error compared to user $U7$ had to spend significantly more amount of time as seen in Figure 7.

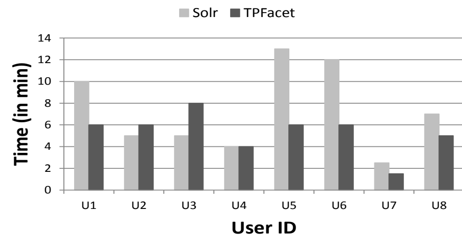

Figure 7: Alternative Search Condition

Figure 7 shows the time users took to finish this task. Statistical analysis shows that TPFacet affects the time taken by ($\chi^2(1) = 2.58,\ p = 0.108$), lowering it by about 2.00 ± 1.14 minutes. Most users were able to do the task 1.5 to 2 times faster using TPFacet as compared to Solr. This task required more time because users had to manually differentiate the IUnits. The main benefit of TPFacet was that users didn't have to try various options using hit-and-trial. They had to look through the IUnits to find the discriminating attribute values, but then it was just trying very few possible alternate choices to see which one gives the best result.

6.3 Performance

Computational time is a crucial constraint for all user facing applications because users expect almost instantaneous response. In this subsection, we evaluate whether TPFacet can provide interactive responses. We performed all our performance experiments on the YahooUsedCar dataset with 40K tuples and 11 attributes. When users browse over e-commerce sites, they rarely deal with result size that is more than 30K-40K tuples and 5-10 queriable attributes. Thus

we evaluate our system using all the tuples of our used-car dataset as query result, with all its attributes being used as queriable attributes.

Our experiments show that TPFacet can give acceptable performance by just using computationally efficient feature selection and clustering algorithms. Each of our experimental graphs are based on average readings of 50 simulations, where for each simulation we generate a different query result by randomly selecting a subset of tuples and/or attributes. The default parameters in these experiments are: the number of Compare Attribute $\mathcal{I} = 11$, the number of generated IUnits $l = 10$, the number of IUnits shown $k = 6$, and the number of attribute-values selected in the Pivot Attribute $\mathcal{V} = 5$. In these experiments, we assume that if the total size of the query result set is $|\mathcal{R}|$, then each attribute value $v \in \mathcal{V}$ has $|\mathcal{R}|/|\mathcal{V}|$ tuples.

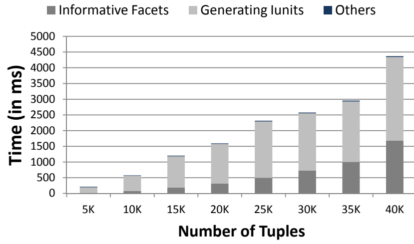

Figure 8: Worst Case System Performance

Figure 8 shows the total time to compute the CAD View for different sizes of query result. In the shown graph, we do not do any optimizations, except using a computationally efficient feature selection and clustering algorithm, that can lead to better system performance. Moreover, we chose system parameter values to demonstrate the worst-case performance of our system. For example, we kept $|\mathcal{I}| = 11$ and $l = 15$. When we consider interaction between many attributes (large $|\mathcal{I}|$) or try to compute many interactions (large l), then it decreases system performance, as shown in later experiments. We divide the total time into three parts: time to compute Compare Attribute, time to generate IUnits and time for all remaining steps, such as top-k ranking, and similarity between IUnits and attribute-values, that we represent collectively as others. We can see that the most computationally intensive parts of TPFacet is computing the top Compare Attribute and generating candidate IUnits. Total time for all other steps is negligible because of the small values of k and $|\mathcal{V}|$ established due to user's display constraint. We can see that even this naive solution is acceptable when the result size is less 15K. But as we increase the result set size, we can see that the time to compute CAD View increases and becomes almost 4.5 secs for 40K tuples. Since the result set size is likely to be the largest in the initial stages of exploration, and since this is also likely to be when the user really needs interactive response to freely try alternatives, a multi-second response time is too slow. To alleviate this problem, we developed several optimizations.

Optimization 1. Sampling — Sampling can improve both feature selection and clustering. For all our attributes, when we computed the set of top ranked Compare Attribute using a small random sample of size 5K-10K, we always got almost the same set, as we got from any larger sample size, including the full dataset. As shown in Figure 8, computing Compare Attribute takes only 20-50 ms for 5K-10K tuples, as compared to 1700 ms for 40K tuples. Quality of Compare Attribute is more crucial when users are towards the end of their exploration, and at that time even the exact computation will take very short time due to small result size. Even if there were some degradation in quality due to sampling, it may not matter much in the intial stages. Similarly, we can also reduce the time for generating IUnits by generating IUnits from a small sample.

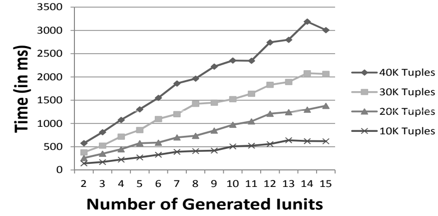

Figure 9: Number of Generated IUnits vs Time

Optimization 2. Varying Generated IUnits — Figure 9 shows the effect of number of generated IUnits l on computation time for different result sizes. We observed that as we increased the number of generated IUnits, it increases computation time due to increased time for clustering. For small 10K result size, computation time is small, less than 500 ms, even when we generate 15 IUnits per attribute value. However, if the result set is large and we generate large number of IUnits, as shown in Figure 8 for 40K tuples with $l = 15$, then it slows down system performance. When users are in their beginning stages of exploration, it is hard to know their preference because their query is too broad. Generating more IUnits and finally ranking is meaningful when we know users' preference more precisely, which typically happens near the end-stages of exploration. Thus we generate fewer IUnits when the result set is very large, so that we can provide a good summary of all options. As users narrow down their exploration, we increase the number of generated IUnits and return better top-k IUnits.

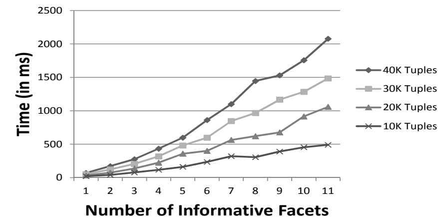

Figure 10: Number of Compare Attributes vs Time

Optimization 3. Fewer Compare Attributes — Figure 10 shows the effect of number of Compare Attributes on computing clusters for different result sizes. As we increase the number of Compare Attributes, it increases computation time because we need to look at the interaction between larger number of attributes. By showing few Compare Attributes we can cluster even 40K tuples in less than 500 ms.

By combining all the above optimizations in creating the CAD View, we can greatly increase the performance of TPFacet system. For example, we can get an CAD View for 40K tuples in less than 500 ms.

7. RELATED WORK

Exploratory search [26, 27, 24, 19] has recently become an important research problem in IR, HCI and database communities. We defined a new exploratory search problem in databases. In evaluating exploratory search systems we cannot separate human behavior from the search system. Since users have diverse background knowledge and information need, it is difficult to evaluate exploratory search systems. Designing evaluation metrics and methodologies for exploratory search system is a challenging research problem [26]. We presented a detailed user-study, based on explicit exploration/understanding tasks with quantitative measures, to evaluate our system.

The CAD View is a summary of important interactions between attributes. Measuring attribute interactions is a part of broader feature selection problem [12, 22, 18] in machine learning. In databases, attribute interactions are often measured in form of functional dependencies [8, 16] and referential integrities. Although standard feature selection can find the interaction between attributes, a Bayesian network [15] can provide a more accurate description of attribute interactions by giving probabilistic dependencies between attributes. These techniques can be used to create CAD Views with other types of data summaries.

Large volumes of relational data are often summarized using data warehousing and OLAP technology [10]. There are also many data mining techniques, such as clustering [20, 3] and decision trees [4, 6], that can group data into meaningful groups according to some user given notion of similarity. A central property of these algorithms is that they depend on the data and are independent of the user's interest. Therefore, the results are often not related to the user's specific exploratory goal. In this paper, we presented a context dependent summarization technique.

Faceted categorization and clustering are both grouping techniques. Hearst [14] presents a nice comparison of how these two techniques complement each other. Various usability studies have shown that users prefer the predictable faceted categorization over clustering [7]. In this paper, we combined faceted browsing with clustering to build the TPFacet system that has benefits of both faceted navigation and clustering.

8. CONCLUSION

In this paper, we presented an exploratory search system for relational databases. Our solution relies on a novel data summarization technique called the CAD View, which provides a context dependent summary of relational result set. We showed through an extensive user study that the CAD View can help users gain quick data familiarity with complex datasets. Although computing the CAD View is computationally intensive, we provided optimizations that enable it to be easily integrated with existing search interfaces, without compromising system performance.

9. REFERENCES

[1] Yahoo used-car. `http://autos.yahoo.com/used_cars.html`.

[2] Apache solr. `http://lucene.apache.org/solr/`, 2014.

[3] P. Berkhin. A survey of clustering data mining techniques. In *GMD*, pages 25–71. Springer, 2006.

[4] K. Chakrabarti, S. Chaudhuri, and S. Hwang. Automatic categorization of query results. In *SIGMOD*, pages 755–766. ACM, 2004.

[5] K. Chen and L. Liu. Clustermap: Labeling clusters in large datasets via visualization. In *CIKM*, pages 285–293. ACM, 2004.

[6] Z. Chen and T. Li. Addressing diverse user preferences in sql-query-result navigation. In *SIGMOD*, pages 641–652. ACM, 2007.

[7] J. C. Fagan. Usability studies of faceted browsing: A literature review. *ITL*, 29(2):58–66, 2013.

[8] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *TKDE*, 23(5):683–698, 2011.

[9] A. Frank and A. Asuncion. UCI Machine Learning Repository, 2010.

[10] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[11] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *ICDE*, pages 512–521. IEEE, 1999.

[12] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *JMLR*, 3:1157–1182, 2003.

[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[14] M. A. Hearst. Clustering versus faceted categories for information exploration. *Communications of the ACM*, 49(4):59–61, 2006.

[15] D. Heckerman. *A tutorial on learning with Bayesian networks*. Springer, 2008.

[16] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, pages 647–658. ACM, 2004.

[17] H. Jagadish and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, pages 275–286, 1998.

[18] I. Kononenko. Estimating attributes: analysis and extensions of relief. In *ECML*, pages 171–182. Springer, 1994.

[19] G. Koutrika, L. V. Lakshmanan, M. Riedewald, and K. Stefanidis. Exploratory search in databases and the web. In *EDBT/ICDT Workshops*, pages 158–159, 2014.

[20] C. Li, M. Wang, L. Lim, H. Wang, and K. Chang. Supporting ranking and clustering as generalized order-by and group-by. *SIGMOD*, 2007.

[21] B. Liu and H. Jagadish. Using trees to depict a forest. *VLDB*, 2009.

[22] H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *TKDE*, 17(4):491–502, 2005.

[23] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.

[24] G. Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.

[25] L. Qin, J. X. Yu, and L. Chang. Diversifying top-k results. *VLDB Endowment*, 5(11):1124–1135, 2012.

[26] R. W. White and R. A. Roth. Exploratory search: Beyond the query-response paradigm. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–98, 2009.

[27] M. L. Wilson, R. W. White, et al. Evaluating advanced search interfaces using established information-seeking models. *ASIST*, 60(7):1407–1422, 2009.

[28] B. Winter. A very basic tutorial for performing linear mixed effects analyses, 2001.

[29] T. Wu, X. Li, D. Xin, J. Han, J. Lee, and R. Redder. DataScope: viewing database contents in Google Maps' way. *VLDB*, 2007.