Smart Drill-Down

Manas Joglekar
Stanford University
manasrj@stanford.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

Aditya Parameswaran
University of Illinois (UIUC)
adityagp@illinois.edu

ABSTRACT

We present a data exploration system equipped with *smart drill-down*, a novel operator for interactively exploring a relational table to discover and summarize "interesting" groups of tuples. Each group of tuples is described by a *rule*. For instance, the rule $(a, b, \star, 1000)$ tells us that there are a thousand tuples with value a in the first column and b in the second column (and any value in the third column). Smart drill-down presents an analyst with a list of rules that together describe interesting aspects of the table. The analyst can tailor the definition of interesting, and can interactively apply smart drill-down on an existing rule to explore that part of the table. The problems underlying smart drill-down are NP-HARD, so our system uses an approximation algorithm for choosing the best list of rules to display. In addition, our system uses a dynamic sampling and memory management scheme to achieve low response times while interacting with large tables.

1. INTRODUCTION

Analysts often use OLAP (Online Analytical Processing) operations such as drill down (and roll up) [2] to explore relational databases. These operations are very useful for analytics and data exploration and have stood the test of time; all commercial OLAP systems in existence support these operations. (Recent reports estimate the size of the OLAP market to be $10+ Billion [4].)

However, there are cases where drill down is ineffective; for example, when the number of distinct values in a column is large, vanilla drill down could easily overwhelm analysts by presenting them with too many results (i.e., aggregates). Further, drill down only allows us to instantiate values one column at a time, instead of allowing simultaneous drill downs on multiple columns—this simultaneous drill down on multiple columns could once again suffer from the problem of having too many results, stemming from many distinct combinations of column values.

In this demonstration, we present a new interaction operator that is an extension to a traditional drill down operator, aimed at providing *complementary* functionality to drill down in cases where drill down is ineffective. We call our operator *smart drill down*. At a high level, smart drill down lets analysts zoom into the more "interesting" parts of a table or a database, with fewer operations, and without having to examine as much data as traditional drill down. Note that our goal is *not* to replace traditional drill down functionality, which we believe is fundamental; instead, our goal is to provide auxiliary functionality which analysts are free to use whenever they find traditional drill downs ineffective.

In addition to this new operator called smart drill down, our system implements novel sampling techniques to compute the results for this operator *in an interactive fashion* on increasingly larger databases. Unlike the traditional OLAP setting, these computations require no pre-materialization, and can be implemented within or on top of any relational database system.

Store	Product	Region	Count	Weight
$\star$	$\star$	$\star$	6000	0

Table 1: Initial summary

Store	Product	Region	Count	Weight
$\star$	$\star$	$\star$	6000	0
$\triangleright$ Target	bicycles	$\star$	200	2
$\triangleright \star$	comforters	MA-3	600	2
$\triangleright$ Walmart	$\star$	$\star$	1000	1

Table 2: Result after first smart drill down

The best way to explain smart drill down is through a simple example.

Example 1. *Consider a table with columns 'Department Store', 'Product', 'Region' and 'Sales'. Suppose an analyst queries for tuples where Sales were higher than some threshold, in order to find the best selling products. If the resulting table has many tuples, the analyst can use traditional drill down to explore it. For instance, the system may initially tell the analyst there are 6000 tuples in the answer, represented by the tuple ($\star$, $\star$, $\star$, 6000, 0), as shown in Table 1. The $\star$ character is a wildcard that matches any value in the database. The Count attribute can be replaced by a Sum aggregate over some measure column, e.g., the total sales. The right-most Weight attribute is the number of non-$\star$ attributes; its significance will be discussed shortly. If the analyst drills down on the Store attribute (first $\star$), then the operator displays all tuples of the form (X, $\star$, $\star$, C, 1), where X is a Store in the answer table, and C is the number of tuples for X (or the aggregate sales for X).*

Instead, when the analyst uses smart drill down on Table 1, he obtains Table 2. The ($\star$, $\star$, $\star$, 6000) tuple is expanded into 3 tuples that display noteworthy or interesting drill downs. The number 3 is a user specified parameter, which we call k.

For example, the tuple (Target, bicycles, $\star$, 200, 2) says that there are 200 tuples (out of the 6000) with Target as the first column value and bicycle as the second. This fact tells the analyst that Target is selling a lot of bicycles. The next tuple tells the analyst that comforters are selling well in the MA-3 region, across multiple stores. The last tuple states that Walmart is doing well in general over multiple products and regions. We call each tuple in Table 2 a rule *to distinguish it from the tuples in the original table that is being explored. Each rule summarizes the set of tuples that are described by it. Again, instead of Count, the operator can display a Sum aggregate, such as the total Sales.*

Say that after seeing the results of Table 2, the analyst wishes to dig deeper into the Walmart tuples represented by the last rule. For instance, the analyst may want to know which states Walmart has more sales in, or which products they sell the most. In this case, the analyst clicks on the Walmart rule, obtaining the expanded sum-

Store	Product	Region	Count	Weight
⋆	⋆	⋆	6000	0
▷ Target	bicycles	⋆	200	2
▷ ⋆	comforters	MA-3	600	2
▷ Walmart	⋆	⋆	1000	1
▷ ▷ Walmart	cookies	⋆	200	2
▷ ▷ Walmart	⋆	CA-1	150	2
▷ ▷ Walmart	⋆	WA-5	130	2

Table 3: Result after second smart drill down

mary in Table 3. The three new rules in this table provide additional information about the 1000 *Walmart tuples. In particular, one of the new rules shows that Walmart sells a lot of cookies; the others show it sells a lot of products in the regions CA-1 and WA-5.*

When the analyst clicks on a rule r, smart drill down expands r into k sub-rules that as a set are deemed to be "interesting." (We discuss other smart drill down operations in Section **??**.*) There are three factors that make a rule set interesting. One is if it contains rules with high Count (or total sales) fields, since the larger the count, the more tuples are summarized. A second factor is if the rules have high weight (number of non-⋆ attributes). For instance, the rule (Walmart, cookies, AK-1,* 200*,* 3*) seems more interesting than (Walmart, cookies, ∗,* 200*,* 2*) since the former tells us the high sales are concentrated in a single region. A third desirability factor is diversity: For example, if we already have the rule (Walmart, ⋆, ⋆,* 1000*,* 1*) in our set, we would rather have the rule (Target, bicycles, ⋆,* 200*,* 2*) than (Walmart, bicycles, ⋆,* 200*,* 2*) since the former rule describes tuples that are not described by the first rule.*

Our system combines these three factors in order to obtain a single desirability score for a set of rules. Our score function can actually be tuned by the analyst (by choosing how weights are computed), providing significant flexibility in what is considered a good set of rules. We also use an efficient optimization procedure to maximize score, invoked by smart drill down to select the set of k rules to display.

Compared to traditional drill down, our smart drill down has two important advantages:

- Smart drill down limits the information displayed to the most interesting k facts (rules). With traditional drill down, a column is expanded and *all* attribute values are displayed in arbitrary order. In our example, if we drill down on say the store attribute, we would see all stores listed, which may be a very large number.
- Smart drill down explores several attributes to open up together, and automatically selects combinations that are interesting. For example, in Table 2, the rule (Target, bicycles, ⋆, 200, 2) is obtained after a single drill down; with a traditional approach, the analyst would first have to drill down on Store, examine the results, drill down on Product, look through all the displayed rules and then find the interesting rule (Target, bicycles, ⋆, 200, 2).

Our work on smart drill down is related to table summarization and anomaly detection [6, 5, 7, 3]. These works mostly focus on giving the most "surprising" information to the user, i.e., information that would minimize the Kullback-Liebler(KL) divergence between the resulting maximum entropy distribution and the actual value distribution. For instance, if a certain set of values occur together in an unexpectedly small number of tuples, that set of values may be displayed to the user. In contrast, our algorithm focuses on rules with high counts, covering as much of the table as possible. Furthermore, our summarization is couched in an interactive environment, where the analyst directs the drill down and can tailor the optimization criteria. Nevertheless, one can envision extending traditional and smart drill down to provide an additional option of anomaly detection.

To reiterate, our chief contribution in this system is the *smart drill down* interaction operator, an extension of traditional drill down, aimed at allowing analysts to zoom into the more "interesting" parts of a dataset. We also develop techniques to support this operator on increasingly larger datasets:

- *Basic Interaction:* Finding the optimal list of rules to display is NP-HARD, so we use an algorithm to find the approximately optimal list of rules to display when the user performs a smart drill down operation.
- *Dynamic Sample Maintenance:* To improve response time on large tables, we build a framework for dynamically maintaining samples in memory to support smart drill down. Optimal identification of samples is once again NP-HARD, so we use an approximate scheme for dynamically maintaining and using multiple samples of the table in memory.

We formally describe our problem in Section 2. Then in Section 3, we describe the components of our system. Finally in Section 4, we outline the demo scenario and describe how users can interact with our system.

2. FORMAL DESCRIPTION

Our system first takes as input a relational table, which we call $\mathcal{D}$. For the purpose of the rest of the discussion, we will operate on this table $\mathcal{D}$. We let T denote the set of tuples in $\mathcal{D}$, and C denote the set of columns in $\mathcal{D}$.

Our objective is to enable smart drill downs on this table or on portions of it: the result of our drill downs are lists of *rules*. A *rule* is a tuple with a value for each column of the table. In addition, a rule has other attributes, such as count and weight associated with it. The value in each column of the rule can either be one of the values in the corresponding column of the table, or ⋆, representing a wildcard character representing all values in the column. A rule r is said to *cover* a tuple t from the table if all non-⋆ values for all columns of the rule match the corresponding values in the tuple. The *Count* of a rule is the number of tuples covered by that rule.

A *rule-list* is an ordered list of rules returned by our system in response to a smart drill down operation. When a user drills down on a rule r to know more about the part of the table covered by r, we display a new rule-list below r. For instance, the second, third and fourth rule from Table 2 form a rule-list, which is displayed when the user clicks on the first (trivial) rule. Similarly, the second, third and fourth rules in Table 3 form a rule-list, as do the fifth, sixth and seventh rules. We now define some additional properties of rules; these properties help us "score" individual rules as part of a rule-list.

There are two portions that constitute our scores for a rule as part of a rule list. The first portion dictates how much the rule r "covers" the tuples in $\mathcal{D}$; the second portion dictates how "good" the rule r is (independent of how many tuples it covers). The reason why we separate the scoring into these two portions is that they allow us to separate the inherent "goodness" of a rule from how much it captures the data in $\mathcal{D}$.

We now describe the first portion: We define *MCount*(r, R) (which stands for 'Marginal Count') as the number of tuples covered by r but not by any rule before r in the rule-list R. A high value of $MCount$ indicates that the rule not only covers a lot of tuples, but also covers parts of the table not covered by previous rules.

Now, onto the second portion: we let W denote a function that assigns a non-negative *weight* to a rule based on how good the rule is, with higher weights assigned to better rules. As we will see, the weighting function does not depend on the specific tuples in $\mathcal{D}$, but could, as we will see later, depend on the number of ⋆s

in r, the schema of $\mathcal{D}$, as well as the number of distinct values in each column of $\mathcal{D}$. A weighting function is said to be *monotonic* if for all rules r_1, r_2 such that r_1 is a sub-rule of r_2, we have $W(r_1) \leq W(r_2)$; we focus on monotonic weighting functions because we prefer rules that are more "specific" rather than those that are more "general" (thereby conveying less information). We further describe our weighting functions in Section **??**.

Thus, the total score for our list of rules is given by

$$\text{Score}(R) = \sum_{r \in R} \underbrace{MCount(r, R)}_{\text{coverage of } r \text{ in } \mathcal{D}} \times \underbrace{W(r)}_{\text{weight of } r}$$

Overall, our goal is to choose the rule-list maximizing total score.

Our smart drill downs still display the Count of each rule rather than the $MCount$. This is because while $MCount$ is useful in the rule selection process, Count is easier for a user to interpret. In any case, it would be a simple extension to display MCount in another column.

Formal Problem: We now formally define our problem:

Problem 1. *Given a table T, a monotonic weighting function W, and a number k, find the list R of k rules that maximizes*

$$\sum_{r \in R} W(r) \times MCount(r, R)$$

for one of the following smart drill down operations:

- [*Rule drill down*] *If the user clicked on a rule r', then all $r \in R$ must be super-rules of r'*
- [*Star drill down*] *If the user clicked on a $\star$ on column c of rule r', then all $r \in R$ must be super-rules of r' and have a non-$\star$ value in column c*

3. SUMMARY OF CONTRIBUTIONS

Our system consists of two main components. The first component determines what rules to display to a user based on the user's latest interaction, and the values of parameters such as number of rules to display (k), weighting function W to use, and so on. In order to do this, it has to make a pass through the table data several times. This can be expensive for big tables, so we dynamically maintain multiple samples of different parts of the table in memory instead. The second component is responsible for maintaining samples in memory and updating them when required. We now describe these two components in further detail below.

The problem of choosing the optimal rule list of a given size, is NP-Hard. However, we find an approximately optimal solution as follows: We first notice that given a set of rules, a rule-list consisting of those rules has the highest score if the rules are sorted in decreasing order by weight. So we can define the score of a rule *set* to be the score of the rule-list obtained by ordering rules of the set in decreasing order by weight. Thus our problem reduces to that of finding the highest scoring rule set. As long as the weight function is monotonic, the score of a rule-set can be shown to be submodular. Then we use the fact that a submodular function can be optimized using a greedy algorithm to choose rules one at a time in a greedy manner until we have k rules. We call the above algorithm BRS (which stands for **B**est **R**ule **S**et). Additional details on our approximation algorithm can be found in our technical report [1].

The second component of our system is called the 'SampleHandler'. It takes as input, the memory capacity M, and a parameter called $minSS$. $minSS$ is a user-specified parameter that determines the sample size required to run

4. DEMO OVERVIEW

5. REFERENCES

[1] https://www.stanford.edu/~manasrj/Papers/SmartDrillDown.pdf.

[2] A. Bosworth, J. Gray, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Technical report, Microsoft Research, 1995.

[3] K. E. Gebaly, P. Agrawal, L. Golab, F. Korn, and D. Srivastava. Interpretable and informative explanations of outcomes. *PVLDB*, pages 61–72, 2014.

[4] R. Kalakota. Gartner: Bi and analytics a $12.2 billion market, july 2013 (retrieved october 30, 2014).

[5] S. Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, pages 307–316, 2000.

[6] S. Sarawagi. User-cognizant multidimensional analysis. *The VLDB Journal*, pages 224–239, 2001.

[7] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *EDBT*, pages 168–182, 1998.

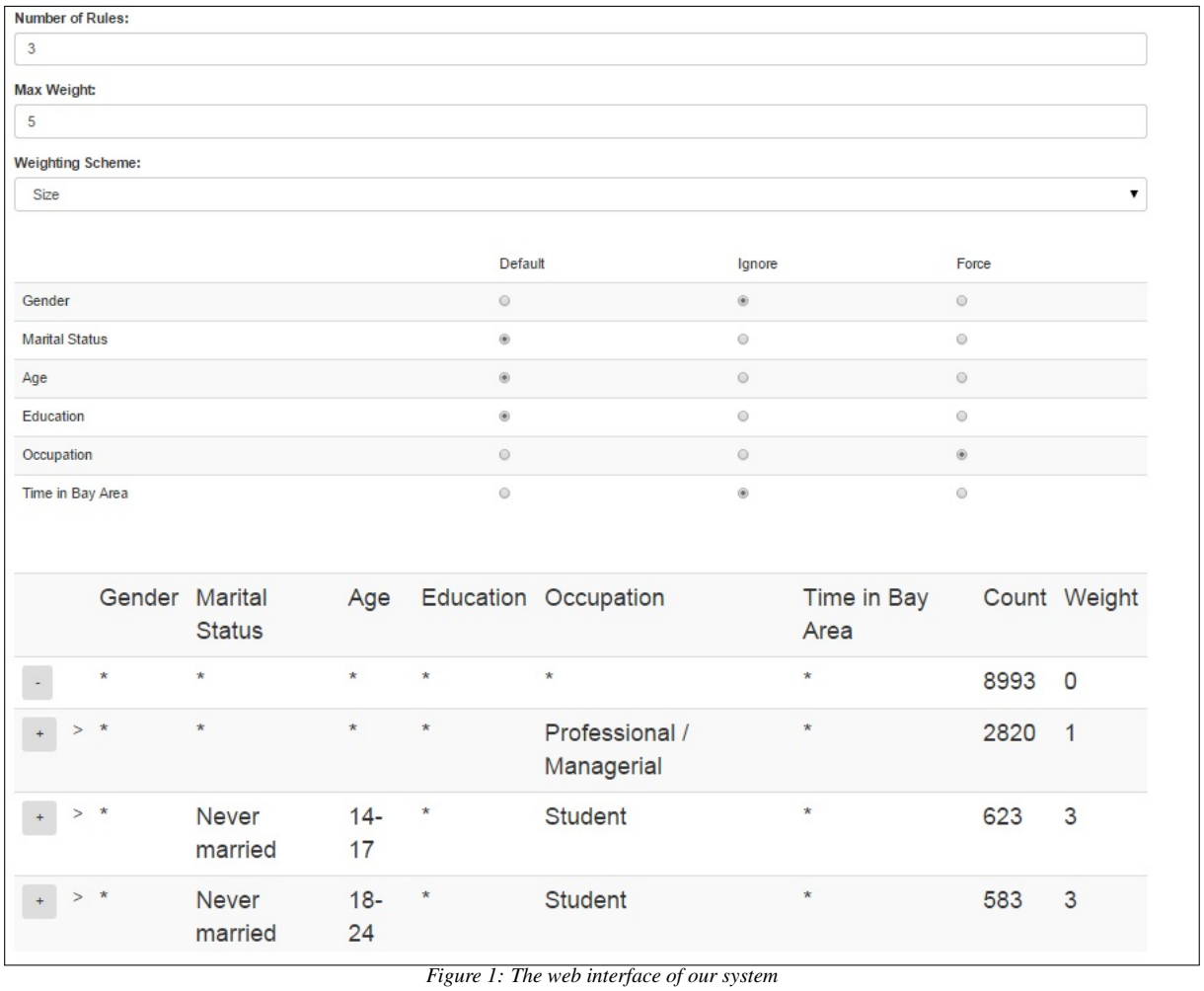

Figure 1: The web interface of our system