HDSampler: Revealing Data Behind Web Form Interfaces

Anirban Maiti Arjun Dasgupta
CSE Dept
UT Arlington
Arlington, TX, USA
{anirban.maiti, arjundasgupta}@uta.edu

Nan Zhang *
CS Dept
George Wash. Univ.
Washington, DC, USA
nzhang10@gwu.edu

Gautam Das †
CSE Dept
UT Arlington
Arlington, TX, USA
gdas@uta.edu

ABSTRACT

A large number of online databases are hidden behind the web. Users to these systems can form queries through web forms to retrieve a small sample of the database. Sampling such hidden databases is widely desired for understanding the nature and quality of data stored in them. We have developed `HDSampler`, which to the best of our knowledge is the first practical system for sampling structured hidden web databases. It enables efficient sampling of the databases and accurate answering of aggregate queries, to provide analysts with valuable information for data analytics, as well as help power a multitude of third-party applications such as web-mashups and meta-search engines. For the purpose of this demo, we present an instance of `HDSampler` on Google Base - a content-rich hidden web database maintained by Google. By using `HDSampler`, the demo reveals a snapshot of the marginal distribution of various attributes of Google Base in a matter of minutes.

Categories and Subject Descriptors

H.2.8 Database applications

General Terms

Algorithms, Design, Measurement, Performance

Keywords

Hidden databases, sampling, top-k interfaces

1. INTRODUCTION

Form-based web interface has become an increasingly popular way for online data providers to furnish data to customers and end users. The objective is to search capabilities of the data to interested users/customers without publishing the database in its entirety. As an example, consider a large car dealership aiming to enable inquiries to its inventory database. Instead of publishing a long list of all inventory vehicles, they may instead provide a web form to which any prospective customer can access using a web browser. The web form allows a customer to construct a search query by choosing desired values for a combination of attributes such as make, model, price range, color, etc., and issue the query to the (back-end) inventory database by clicking the submit button. Then, the web form displays the query answers (e.g., vehicles satisfying the search conditions) returned from the database after applying certain restrictions, such as the maximum number of tuples displayed per query. If the customer finds the results to be too broad or too narrow, s/he can refine or broaden the query by adding or dropping attributes, until the desired results are returned. We call such an interface as the *Conjunctive Web Form Interface*.

The usage of a conjunctive web form interface enables personalized access for individual users, but also poses challenges to analytics over hidden databases. Restrictions applied to the display of query results, especially a commonly used top-k strategy which limits the results to the top-k tuples according to a ranking function, makes it impossible to directly retrieve all tuples by issuing (SELECT *)-like queries. As an increasing number of data providers are adopting the conjunctive web form interface, a great deal of interest has been generated in the research community on bypassing these interfaces to support analytics over the valuable backend data. Crawling these databases has been explored in depth by the information retrieval community, primarily to facilitate the addition of these datasets to a search engine's index. Nonetheless, crawling a very large hidden database can be extremely expensive, and could be impossible when data providers limits the maximum number of queries that can be issued by an IP address.

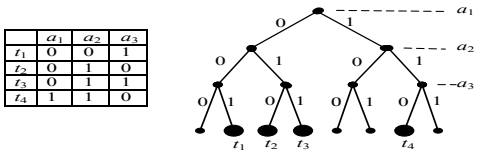
We approach this problem from a different angle and speculate that a tool which provides a *sneak peak* to hidden data statistics would be much more efficient while remaining useful. A data analyst who desires information on the distribution of the data tuples and other aggregate information about the data usually prefers fast but approximate answers over precise answers that require an expensive crawl of the entire database. In the above example, if one wants to learn the percentage of Japanese cars in the dealer's inventory, a very small number of uniform random samples of the underlying database can provide a quite accurate answer. Finding a small representative sample rather than a database crawl can thus be used effectively for numerous analytical and decision-making tasks. It can also help numerous third-party applications such as web-mashups and meta-search engines, which often need to decide on the quality and coverage of the data available at different hidden web sources.

In [1], we proposed HIDDEN-DB-SAMPLER, an algorithm for drawing simple random samples from a hidden database. Based

*Partially supported by NSF grants 0852673, 0852674 and 0845644

†Partially supported by NSF grants 0845644, 0812601 and grants from Microsoft Research and Nokia Research

ACM SIGMOD '09 Providence, RI, USA



	a_1	a_2	a_3
t_1	0	0	1
t_2	0	1	0
t_3	0	1	1
t_4	1	1	0

Figure 1: Query Tree of a Boolean Database

on the algorithm, this demo presents HDSampler, a practical system for sampling databases hidden behind conjunctive web form interfaces. HDSampler provides an exploratory tool for data analysts and domain researchers to quickly draw samples from a hidden database to facilitate decision making tasks. It is a generic tool that can be easily tailored to target various data sources behind conjunctive web form interfaces. A set of input parameters are provided for performance tuning, in particular for a user to make a proper tradeoff between the efficiency of sampling and the skew of samples being drawn.

2. REVIEW OF HIDDEN-DB-SAMPLER

We start our discussion by a brief overview of HIDDEN-DB-SAMPLER. Note that the main challenge to sample hidden databases is that the top-k restriction prevents broad queries from being properly answered i.e., if a query is too broad (i.e., selects more than k tuples), only the top-k tuples will be retrieved according to a ranking function and returned as the query result. The interface will also notify the user that there is an *overflow*, i.e., that not all qualifying tuples are returned. Since the ranking function does not select tuples randomly, a tuple returned by an overflowing query thus cannot be used as a random sample. Such top-k restriction is enforced by many web search interfaces such as Google ($k = 1,000$), MSN Career ($k = 4,000$), Microsoft Solution Finder ($k = 500$), MSN Stock Screener ($k = 25$), and so on.

To address this challenge, the basic idea is to perform a random drill-down starting with extremely broad (and therefore overflowing) queries, and iteratively narrowing it down by adding randomly selected predicates, until a query with fewer than k qualifying tuples is reached. Consider Figure 1 which shows a small Boolean database and a binary tree with 4 levels, where the i-th level represents attribute a_i and the left (resp. right) edge leading out of any internal node is labeled 0 (resp. 1). Thus, each path from root to a leaf represents a specific assignment of Boolean values to attributes, and the leaves represent potential tuples. HIDDEN-DB-SAMPLER essentially performs a random walk down this tree, but makes sure that only paths that lead to actual tuples are explored. In particular, it first executes an under-specified query corresponding to the root node (e.g., "SELECT * FROM D WHERE $a_1 = 0$"). If the query overflows, it is extended by adding a new randomly selected constraint using a_2 (either "$a_2 = 0$" or "$a_2 = 1$"). If it returns 1 to k tuples without overflow, we randomly choose a returned tuple as a sample. This drill-down process leads us to either a legitimate tuple, or an empty result. If the process returns empty, we restart the random walk.

With HIDDEN-DB-SAMPLER, a tuple in a non-overflowing node at higher levels of the tree is more likely to be returned than one at lower levels. Thus, the retrieved tuples are processed by an acceptance-rejection sampling module before being used as samples. This module provides a user an opportunity to make a tradeoff between the uniformity of samples and the efficiency of sampling. A highly uniform sample may take a long time (due to more rejections). However, a set of samples with moderate skew may be obtained quite fast.

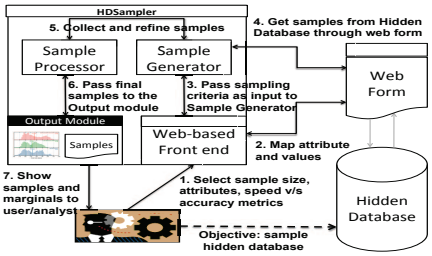


Figure 2: Architecture of the HDSampler

3. HDSAMPLER

Figure 2 depicts the architecture of HDSampler. There are four modules: web-based front end, sample generator, sample processor, and output module. In this section, we introduce the design of each module respectively.

3.1 Web-based Front End

We remind the readers that HDSampler is a generic tool which can be applied to any hidden database with a conjunctive web form interface. To customize HDSampler to a specific data source, one needs to specify the attributes and their domain values required for forming queries.

Demo data source: In the demo, to demonstrate the power of HDSampler on a popularly used system, we customize it to *Google Base* Vehicles database[1], a large online database formed and maintained by Google by integrating numerous vehicle-market data sources and also inputs from individual data providers. The web interface and API for Google Base is a typical conjunctive web form interface which allows a user to retrieve tuples by forming queries on any combination of attributes. The top-k restriction is applied with $k = 1000$. We also bring to attention that the query counts provided by the Google Base interface are approximate estimates generated by some proprietary algorithm and are ignored for the purpose of this system.

Initializing the system: HDSampler allows a variety of settings for users to specify their individual requirements. To begin with, the users have the leeway to add and remove attribute and their value bindings and point HDSampler to either the whole dataset or to a specific selection of attributes. The required number of samples can also be specified. Figure 3 is a screenshot of this feature.

Performance v/s accuracy tradeoffs: Data analysts and users have varying needs. Some want the sampling process to produce results in a matter of minutes in order to make instant decisions. Others may place more importance on obtaining skewless samples with lower emphasis on time. Our system can be configured to meet either need. We provide a slider with one end having the highest efficiency and the other having the lowest skew. The inherent nature of HDSampler dictates a balance between these two parameters and we provides end users the flexibility to make a proper tradeoff according to their needs.

3.2 Sample Generator

The sample generator module forms one part of the core engine of HDSampler with the Sample processor forming the other part. This module is responsible for generating and executing a sequence of random queries according to the HIDDEN-DB-SAMPLER algorithm. It leverages user inputs along with the attributes and their bindings to form queries and submits them to the form interface

[1]http://www.google.com/base/html?a_n0=vehicles&a_y0=9&hl=en&gl=us

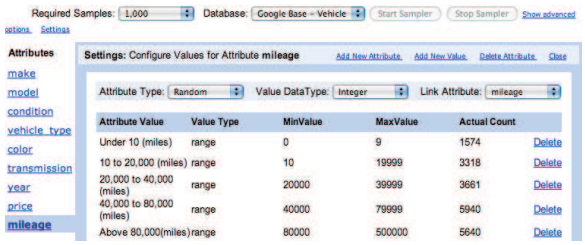


Figure 3: HDSampler attribute settings

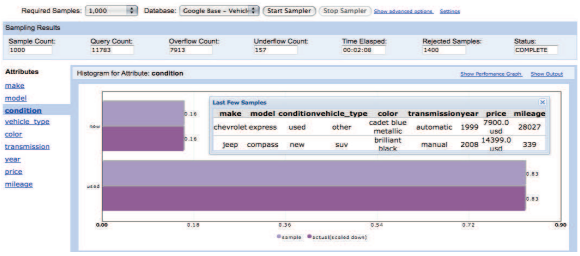


Figure 4: HDSampler showing histograms on the samples

of the data source being sampled. This module returns a set of candidate sample tuples which are then passed on to the Sample Processor. Following an optimization proposed in [2], this module also keeps track of the query history and results to ensure that the random query generation process accumulates savings by not issuing the same query twice, or queries whose results can be inferred from the query history.

3.3 Sample Processor

The candidate samples obtained through the Sample Generator module do not produce the best results in terms of minimizing skew. The sample processor module takes charge of the candidate samples and refines them by applying an acceptance-rejection sampling technique based on the user specified requirement for performance and accuracy. Only a subset of the candidate samples will be included in the output of Sample Processor, while the others are discarded. The final set of samples are stored in this module and are presented to the output module.

3.4 Output Module

The final set of samples found by `HDSampler` can be used for a variety of purposes by users and analysts. The output module presents the end users with a set of these final samples. To target data analysts who wish to gain a quick understanding of the sampled dataset, `HDSampler` generates histograms on the marginal distributions of the attributes and their associated values. This is just one method of representing the sampled dataset with a fixed set of users in mind. The set of samples can be used in a multitude of ways (for example, to execute approximate aggregate queries on a resultant data cube). We provide an interface that allows users to pose aggregate queries (count, sum and average) on a combination of attributes as an example application. This is a simple use case and we encourage users of this system and domain experts to come up with innovative techniques for making sense of the samples in their own settings.

The entire system works in an incremental fashion where the Sample Generator, Sample Processor and Output module generate samples and updates the final sample set and histograms till the desired number of samples are obtained. A kill switch has been included to facilitate stopping the sampling procedure in case the user is satisfied with the samples extracted thus far.

A screenshot of the interface of this module is shown in Figure 4.

Results Validation: In the absence of access to the database being sampled, we resort to verifying our results on Google Base by employing the services of the BRUTE-FORCE-SAMPLER [1] which is proved to produce uniform random samples. However, BRUTE-FORCE-SAMPLER is extremely slow and thus cannot be used in practice for efficient sampling. As proof of concept, we include the marginal counts obtained by a long run of the Brute Force Sampler on Google Base for comparison with the histogram obtained by `HDSampler`.

3.5 Implementation Platform

`HDSampler` has been implemented and hosted on a Xeon Quad Core 2.4 GHz machine running Ubuntu. To enable web access to this system, we used an Apache web server. The front end was designed using a combination of HTML with Javascript (AJAX). AJAX enables the system to give the users a real time feel by presenting seamless updates to the sampling procedure. PHP along with MySQL was used to create the backend engine (Sample Generator and Sample Processor) behind this system.

4. DEMO PLAN

Overview: To facilitate the demo we point `HDSampler` to two data sources: (1) Google Base Vehicles database, and (2) a locally simulated hidden database. Google Base illustrates the use of this system on a candidate real-world hidden database system. The `HDSampler` for Google Base can be started and analyzed over the web using an Internet browser.

Audience Interactions: Our demo allows the audience to use a web interface as shown in Figure 3 to specify the attributes and value bindings of their interest, and adjust various settings related to the efficiency and skewness of sampling. Then, they can specify the number of samples to be collected and start the sampler. The audience can also request the display of attribute histograms they are interested in, as shown in Figure 4. They can also observe the recently collected samples, and stop the sampler at anytime.

A demonstration video is included as a supplemental material.

Backup Plan: In case of low connectivity at the demonstration venue, we have also created a local implementation of the system on a simulated hidden data source. This can be run on a stand-alone system. The local hidden database also allows us to demonstrate the effectiveness of the sampler since this time around the entire dataset can be accessed for validation.

5. SUMMARY

In this demo, we present `HDSampler`, a practical system for the efficient sampling of structured hidden web databases via their proprietary front-end search interfaces. Such samples can be very effective for answering of aggregate queries, to provide analysts with valuable information for data analytics, as well as help power a multitude of third-party applications such as web-mashups and meta-search engines. We demonstrate a customized `HDSampler` for Google Base, and evaluate its effectiveness in terms of accuracy of estimating marginal distribution and efficiency of drawing random samples.

6. REFERENCES

[1] A. Dasgupta, G. Das, and H. Mannila. A random walk approach to sampling hidden databases. *SIGMOD*, 2007.

[2] A. Dasgupta, N. Zhang, and G. Das. Leveraging count information in sampling hidden databases. *ICDE*, 2009.