

# Unbiased Estimation of Size and Other Aggregates Over Hidden Web Databases

Arjun Dasgupta  
University of Texas at Arlington  
arjundasgupta@uta.edu

Xin Jin  
George Washington University  
xjin@gwmail.gwu.edu

Bradley Jewell  
University of Texas at Arlington  
bradley.jewell@uta.edu

Nan Zhang<sup>\*</sup>  
George Washington University  
nzhang10@gwu.edu

Gautam Das<sup>†</sup>  
University of Texas at Arlington  
gdas@uta.edu

## ABSTRACT

Many websites provide restrictive form-like interfaces which allow users to execute search queries on the underlying hidden databases. In this paper, we consider the problem of estimating the size of a hidden database through its web interface. We propose novel techniques which use a small number of queries to produce unbiased estimates with small variance. These techniques can also be used for approximate query processing over hidden databases. We present theoretical analysis and extensive experiments to illustrate the effectiveness of our approach.

## Categories and Subject Descriptors

H.2.7 [Database Administration]; H.3.5 [Online Information Services]: Web-based services

## General Terms

Algorithms, Measurement, Performance

## Keywords

Hidden Databases, Aggregate Query Processing

## 1. INTRODUCTION

In this paper, we develop novel techniques to answer various types of aggregate queries, such as database size, over hidden web databases. Our techniques are *efficient* and provide estimates with *small error*. Most importantly, our estimations are *unbiased*, which none of the existing non-crawling techniques can achieve.

**Hidden databases:** Hidden databases are widely prevalent on the web. They feature restrictive form-like interfaces which allow users

<sup>\*</sup>Partially supported by NSF grants 0852673, 0852674, 0845644 and 0915834 and a GWU Research Enhancement Fund.

<sup>†</sup>Partially supported by NSF grants 0845644, 0812601 and 0915834 and grants from Microsoft Research and Nokia Research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.  
Copyright 2010 ACM 978-1-4503-0032-2/10/06 ...\$10.00.

to form a search query by specifying the desired values for one or a few attributes, and the system returns a small number of tuples satisfying the user-specified selection conditions. Due to limitations of a web interface, the number of returned tuples is usually restricted by a top- $k$  constraint - when more than  $k$  tuples in the database match the specified condition, only  $k$  of them are preferentially selected by a ranking function and returned to the user.

**Problem Definition:** The problem we consider in this paper is how to use the web interface to estimate the size and other aggregates of a hidden database:

- Estimating the number of tuples in a hidden database is by itself an important problem. Many hidden databases today advertise their (large) sizes on public venues to attract customers. However, the accuracy of such a published size is not (yet) verifiable, and sometimes doubtful, as the hidden database owners have the incentive to exaggerate their sizes to attract access. Furthermore, many hidden databases, such as the job-hunting monster.com, do not publicize their total sizes, while such information can be useful to the general public as an economic indicator for monitoring job growth.
- More generally, the ability to approximately answer aggregate queries can enable a wide range of third-party data analytics applications over hidden databases. For example, aggregates may reveal the quality, freshness, content bias and size of a hidden database, and can be used by third-party applications to preferentially select a hidden database with the best quality over other hidden databases.

**Challenges:** A simple approach to obtain the size of a hidden database is to crawl all tuples and then count them. A number of techniques have been proposed for the crawling of hidden databases [2, 20, 25]. This approach, however, requires an extremely large amount of queries to be issued through the web interface. Such a high query cost is infeasible in practice because most hidden databases impose a per-user/IP limit on the number of queries one can issue. For example, Yahoo! Auto, a popular hidden database, has a limit of 1,000 queries per IP address per day.

*Capture-recapture* is another approach extensively studied in the field of statistics for population size estimation [3], and has been recently used to estimate the size of a search engine's corpus [9, 22, 28]. This approach is built upon the sampling of a population, and estimates the population size according to the recaptures counts of entities being sampled. In particular, for a population of  $m$  entities, the capture-recapture approach requires a sample of size  $\Omega(\sqrt{m})$  to form a reasonable estimation.

However, applying capture-recapture over the existing sampling

techniques for hidden databases leads to two problems, on estimation error and query cost, respectively. First, the estimations generated this way are biased and may have high variance<sup>1</sup>. Not only is the capture-recapture approach in general known to produce biased estimations with high variance for large populations [3], but all underlying sampling techniques in the literature for hidden databases are also biased with the bias unknown [13, 14]<sup>2</sup>. For many data analytics applications (e.g., to fairly compare the hidden database size of competing providers), however, the unbiasedness of an estimator is a requirement that cannot be waived. In addition, the estimation variance must be clearly understood and minimized to derive a meaningful confidence interval for the estimation. This cannot be achieved by directly applying capture-recapture over existing hidden database sampling techniques.

Furthermore, to enable an accurate estimation, capture-recapture requires an intolerably large number of queries to be issued. This is due to a distinct challenge for sampling hidden databases, i.e., the significant difference between the size of the database and the set of all possible queries [13, 14]. Such a drastic difference stands in contrast to sampling a search engine’s corpus, for which a common assumption is the existence of a (reasonably small) pool of queries that recall almost all documents [5, 7, 8]. Due to such drastic difference, each of the  $\Omega(\sqrt{m})$  sample tuples requires a small but non-negligible number of queries to generate. As a result, the capture-recapture approach requires a very large query cost over hidden databases which renders it impractical.

Similar to the challenges for estimating the hidden database size, significant obstacles are present for estimating aggregates over a hidden database. The existing sampling-based techniques are not designed to answer aggregate queries, but to sample all tuples with equal probability. Thus, while these techniques may support an estimation of AVG queries, they cannot answer SUM or COUNT queries. Furthermore, even when the precise database size is given, one still cannot generate unbiased estimates from these techniques because the sampling is performed with a biased selection probability distribution over all tuples, and moreover the bias is unknown.

**Outline of Technical Results:** In this paper we initiate a study of estimating, without bias, the size and other aggregates over a hidden database. For size estimation, our main result is HD-UNBIASED-SIZE, an unbiased estimator with provably bounded variance. For estimating other aggregates, we extend HD-UNBIASED-SIZE to HD-UNBIASED-AGG which produces unbiased estimations for aggregate queries.

HD-UNBIASED-SIZE is based on performing random walks over the query space, by starting with a query with very broad selection conditions, and drilling down by adding random conjunctive constraints to the selection condition, until the query selects at most  $k$  tuples. It features three key ideas: *backtracking*, *weight adjustment*, and *divide-&-conquer*. Backtracking enables the unbiasedness of estimation. Weight adjustment and divide-&-conquer both reduce the estimation variance, with divide-&-conquer delivering the most significant reduction for real-world hidden databases.

Backtracking applies when the random walk hits an empty query. Instead of completely restarting the walk, we backtrack to the previous query and attempt adding another constraint, in order to ensure that each trial of the random walk ends with a non-empty

<sup>1</sup>A random estimator  $\hat{\theta}$  for an aggregate  $\theta$  is considered to be *biased* if  $E[\hat{\theta}] \neq \theta$ . The *mean squared error* of the estimator is defined as  $E[(\hat{\theta} - \theta)^2]$ , which is the same as the variance of  $\hat{\theta}$  for an unbiased estimator. It is usually desirable for estimators to be unbiased and have small variance.

<sup>2</sup>Excluded from consideration here are sampling techniques designed under the assumption that the hidden database truthfully disclose its size e.g., [14].

query. The key implication of backtracking is that it enables the precise computation of the selection probability for the returned tuples. As a result, HD-UNBIASED-SIZE is capable of generating an unbiased estimation for the database size.

We note that there is a key difference between backtracking-enabled random walks and the existing sampling techniques over hidden databases. The existing techniques aim to produce uniform random samples but eventually fall short with unknown bias. In comparison, our random walk intentionally produces biased samples, but the bias of the sample is precisely known. As a result, our estimation technique is capable of completely correcting the sampling bias and producing unbiased estimations of database size and other aggregates.

While a backtracking-enabled random walk produces no bias, the variance of its estimation may be large when the underlying data distribution is highly skewed. The objective of weight adjustment is to reduce the estimation variance by “aligning” the selection probability of tuples in the database to the distribution of measure attribute (to be aggregated). For our purpose of estimating the database size, the measure attribute distribution is uniform (i.e., 1 for each tuple). Thus, we adjust the transitional probability in the random walk based on the density distribution of “pilot” samples collected so far. After weight adjustment, each random walk produces an unbiased estimate with gradually reduced variance.

While weight adjustment has the estimation variance converging to 0, the convergence process may be slow for a database that is much smaller than its domain size (i.e., the set of all possible tuples). Divide-&-conquer is proposed to address this problem by carefully partitioning the database domain into a large number of subdomains, such that the vast majority of tuples belong to a small number of subdomains. Then, we perform random walks over certain subdomains and combine the results for estimation of the database size. The reduced size mismatch between the (sub-) query space and the database significantly reduces the final estimation variance, while only a small number of subdomains need to be measured, leading to very small increase on query cost.

A major contribution of this paper is also a theoretical analysis of the quantitative impact of the above ideas on reducing variance. We also describe a comprehensive set of experiments that demonstrate the effectiveness of HD-UNBIASED-SIZE and HD-UNBIASED-AGG over both synthetic and real-world datasets, including experiments of directly applying these algorithms over the web interface of a popular hidden database websites, *Yahoo! Auto*.

In summary, the main contributions of this paper are as follows:

- We initiate the study of *unbiased* estimation of the size and other aggregates over a hidden database through its restrictive web interface.
- We propose a backtracking-enabled random walk technique to estimate hidden database size and other aggregates without bias. To the best of our knowledge, this is the first time an aggregate can be estimated without bias over a hidden database.
- We also propose two other techniques, weight adjustment and divide-&-conquer, to reduce the estimation variance.
- We combine the three techniques to produce HD-UNBIASED-SIZE, an efficient and unbiased estimator for the hidden database size. Similarly, we propose HD-UNBIASED-AGG which supports various aggregate functions and selection conditions.
- We provide a thorough theoretical analysis and experimental studies that demonstrate the effectiveness of our proposed approach over real-world hidden databases.

**Paper Organization:** The rest of this paper is organized as fol-

lows. In Section 2 we introduce preliminaries and discuss simple but ineffective algorithms for aggregate estimation over hidden databases. Sections 3 and 4 are devoted to the development of HD-UNBIASED-SIZE, focusing on achieving unbiasedness and reducing variance, respectively. In Section 5 we discuss the parameter settings for HD-UNBIASED-SIZE and extend it to HD-UNBIASED-AGG. Section 6 contains a detailed experimental evaluation of our proposed approaches. Section 7 discusses related work, followed by conclusion in Section 8.

## 2. PRELIMINARIES

### 2.1 Models of Hidden Databases

We restrict our discussion in this paper to categorical data - we assume that numerical data can be appropriately discretized to resemble categorical data, and exclude tuples with null values from consideration. Consider a hidden database table  $D$  with  $m$  tuples  $t_1, \dots, t_m$  and  $n$  attributes  $A_1, \dots, A_n$ . We assume no duplicate tuple exists in  $D$ . Let  $Dom(\cdot)$  be a function that returns the domain of one or more attributes. As such,  $Dom(A_i)$  represents the domain of  $A_i$ , and  $Dom(A_i, A_j)$  represents the Cartesian product of the domains of  $A_i$  and  $A_j$ .  $|Dom(A_i)|$  represents the cardinality of  $Dom(A_i)$ , i.e., the number of possible values of  $A_i$ .

The table is only accessible to users through a web-based interface. We assume a prototypical interface where users can query the database by specifying values for a subset of attributes. Thus a user query  $q$  is of the form:

```
SELECT * FROM D WHERE Ai1 = vi1 & ... & Ais = vis,
where vij is a value from Domij.
```

Let  $Sel(q)$  be the set of tuples in  $D$  that satisfy  $q$ . As is common with most web interfaces, we shall assume that the query interface is restricted to only return  $k$  tuples, where  $k \ll m$  is a predetermined small constant (such as 10 or 50). Thus,  $Sel(q)$  will be entirely returned iff  $|Sel(q)| \leq k$ . If the query is too broad (i.e.,  $|Sel(q)| > k$ ), only the top- $k$  tuples in  $Sel(q)$  will be selected 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 tuples satisfying  $q$  can be returned. At the other extreme, if the query is too specific and returns no tuple, we say that an *underflow* occurs - i.e., the query is empty. If there is neither overflow nor underflow, we have a *valid* query result. Without causing confusion, we also use  $q$  to represent the set of tuples returned by  $q$ . Note that the number of returned tuples  $|q| = \min(k, |Sel(q)|)$ .

For the purpose of this paper, we assume that a restrictive interface does not allow users to “scroll through” the complete answer  $Sel(q)$  when  $q$  overflows. Instead, the user must pose a new query by reformulating some of the search conditions. This is a reasonable assumption because many real-world top- $k$  interfaces (e.g., Google) only allow “page turns” for limited (e.g., 100) times.

**A running example:** Table 1 depicts a simple table which we shall use as a running example throughout this paper. There are  $m = 6$  tuples and  $n = 5$  attributes, including four Boolean ( $A_1, \dots, A_4$ ) and one categorical ( $A_5 \in [1, 5]$ , only 1 and 3 appear in the table).

### 2.2 Performance Measures

We consider the estimation of aggregate queries with conjunctive conditions of the form  $SELECT\ AGGR(A_j)\ FROM\ D\ WHERE\ A_{i_1} = v_{i_1} \& \dots \& A_{i_s} = v_{i_s}$  where  $AGGR$  is the aggregate function. For example, such an aggregate query might be the number of tuples in the database which we focus on for most part of the paper. It may also be the SUM of prices for all inventory cars of a car dealership’s hidden database (i.e., the inventory balance).

**Table 1: Example: Input Table**

	$A_1$	$A_2$	$A_3$	$A_4$	$A_5$
$t_1$	0	0	0	0	1
$t_2$	0	0	0	1	1
$t_3$	0	0	1	0	1
$t_4$	0	1	1	1	1
$t_5$	1	1	1	0	3
$t_6$	1	1	1	1	1

An aggregate estimation algorithm for hidden databases should be measured in terms of estimation accuracy and query efficiency:

- **Estimation Accuracy:** The mean squared error of an estimator is a composition of its bias and variance. Consider an estimator  $\tilde{\theta}$  for an aggregate  $\theta$ . Then  $MSE(\tilde{\theta}) = E[(\tilde{\theta} - \theta)^2] = E[(\tilde{\theta} - E(\tilde{\theta}))^2] + (E(\tilde{\theta}) - \theta)^2 = Var(\tilde{\theta}) + Bias^2(\tilde{\theta})$ , where  $E[\cdot]$  represents expected value. The task is to design unbiased estimators with minimum variance.
- **Query Efficiency:** Many hidden databases impose limits on the number of queries from a user. The task is to minimize the number of queries issued through the web interface to achieve a given level of estimation accuracy.

### 2.3 Simple Aggregate-Estimation Algorithms

**BRUTE-FORCE-SAMPLER:** This algorithm randomly composes a fully-specified query  $A_1 = v_1, \dots, A_n = v_n$  by generating each  $v_i$  from  $Dom(A_i)$  uniformly at random. There are two possible outcomes: either underflow or valid. After repeating this process for  $h$  times, let  $h_V$  be the number of tuples found. Then, one can estimate the database table size as  $\tilde{m} = |Dom(A_1, \dots, A_n)| \cdot h_V / h$ . It is easy to see that this process will produce an unbiased estimate. Other aggregates can be estimated without bias in a similar way. However, BRUTE-FORCE-SAMPLER is extremely inefficient because the probability for a fully-specified query to return valid is extremely small (i.e.,  $m \ll |Dom(A_1, \dots, A_n)|$ ) [13].

**CAPTURE-&-RECAPTURE:** The Lincoln-Petersen model [27] is a well-known capture-&-recapture estimator for the size of a closed population. Consider the case where we collect two samples,  $C_1$  and  $C_2$ , of a hidden database. The Lincoln-Petersen estimator gives  $\tilde{m} = |C_1| \cdot |C_2| / |C_1 \cap C_2|$  where  $|C_i|$  is the number of tuples in  $C_i$  and  $|C_1 \cap C_2|$  is the number of tuples that appear in both samples. The estimation tends to be positively biased [3]. One can see that the estimator only works when each sample includes (at least)  $\Omega(\sqrt{m})$  tuples, which leads to an extremely expensive process for hidden databases.

### 2.4 Hidden Database Sampling

The basic idea behind the HIDDEN-DB-SAMPLER [13] and its extension HYBRID-SAMPLER [14] is to perform a random drill-down starting with extremely broad (and thus overflowing) queries, and iteratively narrowing it down by adding randomly selected predicates, until a valid query is reached. In particular, consider a Boolean database. The sampler starts with query  $SELECT\ *\ FROM\ D$ . If the query overflows, it is expanded by adding a randomly selected predicate on  $a_1$  (either “ $a_1 = 0$ ” or “ $a_1 = 1$ ”). If the expanded query still overflows, we further expand it by adding random predicates for  $a_2, \dots, a_n$  respectively one at a time. This random walk process leads us to either a valid or an underflowing query. If it reaches an underflow, we restart the random walk. If the process reaches a valid query, we randomly choose a returned tuple and include it in the sample. Since this random walk process more

likely chooses tuples returned by “short” valid queries, we have to apply *rejection sampling* at the end of the random walk. In particular, a sample tuple is rejected with probability  $1 - C/2^h$ , where  $C$  is a pre-determined parameter and  $h$  is the number of predicates in the query from which the tuple is retrieved.

## 2.5 Table of Notations

$D$	hidden database
$k$	maximum number of tuples returned
$t_1, \dots, t_m$	set of tuples in $D$
$A_1, \dots, A_n$	set of attributes in $D$
$Sel(q)$	tuples in $D$ that satisfy the selection conditions of $q$
$ Dom $	domain size of the Cartesian product of all attributes
$ Dom(\cdot) $	domain size of the Cartesian product of selected attributes
$\Omega_{TV}$	set of all top-valid nodes
$p(q)$	probability of $q$ being selected in a drill down
$r$	number of drill downs performed over each subtree
$D_{UB}$	maximum subdomain size for each subtree
$s^2$	estimation variance

## 3. UNBIASEDNESS FOR SIZE ESTIMATION

In this section, we develop the main ideas that enable the *unbiasedness* of estimation for the size of a hidden database.

### 3.1 Random Drill-Down With Backtracking

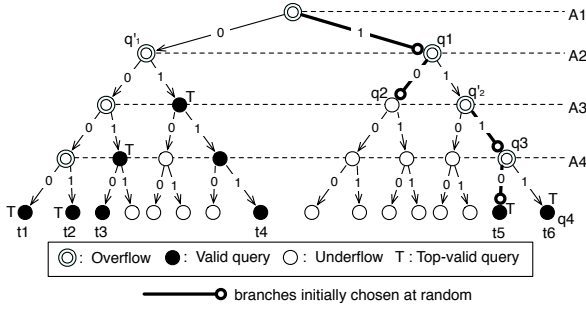


Figure 1: Query Tree for the Running Example

We start with a hidden database with all Boolean attributes, and extend this scenario to categorical attributes in the next subsection. To understand the idea, consider a query tree as depicted in Figure 1 (for the first four Boolean attributes of the running example in Table 1). Each level of the tree represents an attribute, while each outgoing edge from a level represents a possible value of that attribute. Thus, each node represents a conjunctive query defined by the AND of predicates corresponding to the edges from the root to this node. For example,  $q_2$  in Figure 1 represents  $SELECT * FROM D WHERE A_1 = 1 AND A_2 = 0$ . We call a node overflowing, valid, or underflowing according to the result of its corresponding query. Figure 1 shows the class of each node when  $k = 1$ . We also introduce the definition of a *top-valid* node:

**DEFINITION 1. [Top-Valid Query]** Given a query tree, a valid query is *top-valid* iff its parent is overflowing.

All top-valid queries in Figure 1 are marked with symbol T. One can see from this definition that each tuple belong to one and only one top-valid node. For example, Figure 1 has 6 top-valid queries corresponding to 6 tuples as  $k = 1$ .

The main process of BOOL-UNBIASED-SIZE is a random procedure which samples top-valid nodes with certain probability distribution. If we know the probability  $p(q)$  for each top-valid node

$q$  to be chosen, then for any sample node  $q$ , we can generate an unbiased estimation for the database size as  $|q|/p(q)$ , where  $|q|$  is the number of tuples returned by  $q$ .

Following this basic idea, BOOL-UNBIASED-SIZE consists of two steps: The first is a random drill-down process over the query tree to sample a top-valid node. The second is to compute  $p(q)$  for the sampled node. The entire process can be repeated for multiple times to reduce the variance of estimation. Both steps may require queries to be issued through the interface of the hidden database. We describe the two steps respectively as follows, with the pseudocode summarized in Figure 2. Note that the two steps are interleaved in the pseudocode.

---

```

1:  $q \leftarrow$  root node.  $p \leftarrow 1$ .  $i \leftarrow 1$ .
2: Randomly generate  $v \in \{0, 1\}$ .
3: Issue  $q' \leftarrow q \wedge (A_i = v)$ .  $\triangleright$  for Step 1 (random drill-down)
4: if  $q'$  underflows then
5:    $q \leftarrow q \wedge (A_i = 1 - v)$ . Goto 2.  $\triangleright$  Backtracking
6: else if  $q'$  overflows then
7:   Issue  $q \wedge (A_i = 1 - v)$ .  $\triangleright$  for Step 2 (computing  $p(q)$ )
8:   if  $q \wedge (A_i = 1 - v)$  is nonempty then
9:      $p \leftarrow p/2$ .  $\triangleright$  Update  $p(q)$ 
10:  end if
11:   $q \leftarrow q'$ .  $i \leftarrow i + 1$ . Goto 2.
12: end if
13: return  $\tilde{m} \leftarrow |q|/p$ .  $\triangleright$  Return an estimation for database size

```

---

Figure 2: BOOL-UNBIASED-SIZE

The random drill-down process starts from the root node. We choose a branch uniformly at random and issue the corresponding query  $q$ . There are three possible outcomes:

- If  $q$  overflows, we further drill down the tree by selecting each branch of  $q$  with equal probability.
- If  $q$  is valid, i.e., it returns  $|q|$  ( $1 \leq |q| \leq k$ ) tuples without an overflow flag, then we conclude the random walk.
- If  $q$  underflows, then we *backtrack* by considering  $q'$ , the sibling of  $q$  - i.e., the node that shares the same parent with  $q$ . For example, the sibling of  $q_2$  in Figure 1 is  $q'_2$ . Note that the parent of  $q$  must be overflowing because otherwise the drill-down process will terminate before reaching  $q$ . Thus,  $q'$  must overflow. We randomly choose a branch coming out of  $q'$  and follow it to further drill down the tree.

One can see that a drill-down process always terminates with a top-valid node. An example of a random drill-down is shown in Figure 1 with bold edges representing the branches that were initially chosen at random. In this example, backtracking happens when  $q_2$  was chosen - since  $q_2$  overflows, we backtrack and continue drilling down from its sibling node  $q'_2$ . Note that with backtracking, a top-valid node like  $q_4$  may be reached under multiple possibilities of the initially chosen branches. For example, when  $q_1, q'_2, q_3, q_4$  were initially chosen,  $q_4$  would also be reached by the drill-down.

We now consider the second step which computes  $p(q)$ . Note that there is a unique path from the root node to a top-valid node  $q$  - for a random drill-down to reach  $q$ , although edges on this path might not be initially chosen, they must be finally followed (after backtracking). For example, such a path for  $q_4$  in Figure 1 goes through  $q_1, q'_2, q_3$  in order. For each edge of the path, there are only two possible scenarios for choosing between the edge and its sibling (from which the drill-down must choose one to follow):

- *Scenario I*: Both branches are non-underflowing. In this case, each branch is chosen with 50% probability. The selection between  $q_1$  and  $q'_1$  in Figure 1 is an example of this scenario.
- *Scenario II*: One branch is overflowing while the other is underflowing. In this case, the overflowing branch will always be chosen. The selection between  $q_2$  and  $q'_2$  in Figure 1 is an example of this scenario, and  $q'_2$  is always chosen.

In order to compute  $p(q)$ , we must know which scenario occurs for each level from the root to  $q$ . This may or may not require additional queries to be issued. In particular, no additional query is needed for the last level before reaching  $q$ , as Scenario I always occurs there. No query is needed for levels when backtracking applies either, because one branch must underflow. However, for any other level, we must issue the sibling query to determine the correct scenario. Consider the bold edges in Figure 1. Since  $q_1$  was selected during random drill-down, we do not know whether  $q'_1$  is underflowing (Scenario I) or not (II). Thus, we must now issue  $q'_1$  to determine which scenario actually occurred.

After learning the number of occurrences for Scenarios I and II, we can then compute  $p(q)$ . For a query  $q$  with  $h$  predicates, let  $h_1$  and  $h - h_1$  be the number of occurrences for Scenario I and II, respectively. We have  $p(q) = 1/2^{h_1}$ . For example,  $q_4$  in Figure 1 has  $h_1 = 2$  (i.e., while issuing  $q_1$  and  $q_4$ ) and  $p(q) = 1/4$ , leading to an estimation of  $|q|/p(q) = 4$ . Note that this estimation is an instantiation of the Horvitz-Thompson Estimator [19]. The following theorem shows its unbiasedness:

**THEOREM 1. [Estimate (un-)Biasness]** *The estimate generated by the drill-down process is unbiased, i.e., its expected value taken over the randomness of  $q$  is*

$$E \left[ \frac{|q|}{p(q)} \right] = m. \quad (1)$$

**PROOF.** Let  $\Omega_{TV}$  be the set of all top-valid nodes in the tree. Since each tuple belongs to one and only one top-valid node, we have  $\sum_{q \in \Omega_{TV}} |q| = m$ . Since the random drill-down process always terminates on a top-valid query,

$$E \left[ \frac{|q|}{p(q)} \right] = \sum_{q \in \Omega_{TV}} p(q) \cdot \frac{|q|}{p(q)} = m$$

□

### 3.2 Smart Backtracking for Categorical Data

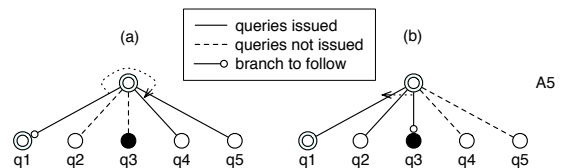
We now discuss how to apply unbiased size estimation to a categorical database. Two changes are required:

- First, Boolean attributes ensure that the sibling of an underflowing node always overflow. There is no such guarantee for categorical databases. Thus, to successfully backtrack from an underflowing branch, we must find one of its sibling branches that returns non-empty *and* count the number of such non-empty siblings (in order to compute  $p(q)$ ). Note that a non-empty branch always exists given an overflowing parent node. A simple backtracking approach is to query all branches to find the (COUNT of) non-empty ones, and then randomly choose a non-empty branch to follow.
- The other change required is the computation of  $p(q)$ . If the above-mentioned simple backtracking is used, the computation of  $p(q)$  becomes  $p(q) = 1/\prod_{i=1}^{h-1} c_i$ , where  $c_i$  is the number of non-underflowing branches for the  $i$ -th predicate en route to the top-valid query  $q$ .

The two changes for categorical databases do not affect the unbiasedness of the estimation as the proof of Theorem 1 remains unchanged. These changes, however, do affect the query cost of the drill-down process. In particular, if the above simple backtracking technique is used, we must issue queries corresponding to all branches to find the number of non-empty ones, leading to a high query cost for large-fanout attributes.

To reduce such a high query cost, we develop *smart backtracking* which aims to avoid testing all branches of a high fanout attribute. Consider a categorical attribute  $A_i$  with possible values  $v_1, \dots, v_w$  ( $w = |Dom(A_i)|$ ). Assume a total order of the values which can be arbitrarily assigned for cardinal or ordinal attributes. In the following, we describe the random drill-down and the computation of  $p(q)$  with smart backtracking, respectively.

To choose a branch of  $A_i$  for further drill down, we first randomly choose a branch  $v_j$ . Backtracking is required if  $v_j$  underflows. With smart backtracking, we test the right-side neighbors of  $v_j$  (in a circular fashion), i.e.,  $v_{(j \bmod w)+1}$ ,  $v_{((j+1) \bmod w)+1}$ , etc, in order until finding one that does not underflow, and follow that branch for further drill down. Figure 3 (a) demonstrates the branches for the categorical attribute  $A_5$  in the running example with  $w = 5$ . To backtrack from  $q_4$ , we test  $q_5$  and then  $q_1$ . Since  $q_1$  returns nonempty, we follow branch  $q_1$  for further drill-down.



**Figure 3: An Example of Smart Backtracking**

We now consider the computation of  $p(q)$ . With the new strategy, the probability for choosing a non-empty branch  $v_j$  is  $(w_U(j) + 1)/w$ , where  $w_U(j)$  is the number of consecutive underflowing branches immediately preceding  $v_j$  (again in a circular fashion). For example, in Figure 3,  $q_1$  and  $q_5$  have  $w_U = 2$  and 1, respectively. To compute  $w_U(j)$ , we need to issue queries corresponding to the left-side neighbors of  $v_j$ , i.e.,  $v_{(j-2) \bmod w+1}$ ,  $v_{(j-3) \bmod w+1}$  etc, in order until finding a non-empty branch. Then, we learn  $w_U(j)$  and is able to compute the probability of following  $v_j$ . Figure 3 (a) and (b) shows the queries one needs to issue after committing to branch  $q_1$  and  $q_3$ , respectively.

With smart backtracking, for a given node, the expected number of branches one needs to test is

$$QC = 1 + \sum_{j=1}^w \frac{(w_U(j) + 1)^2}{w}, \quad (2)$$

where  $w_U(j) = -1$  if  $v_j$  is an empty branch. For example,  $QC = 3.6$  for  $A_5$  in Figure 3. For attributes with larger fanouts, smart backtracking may reduce the query cost more significantly.

### 3.3 Discussion

#### 3.3.1 Comparison with Prior Samplers

To illustrate the effectiveness of our backtracking-enabled random drill-down approach, we compare it with two prior algorithms discussed in the preliminary section: BRUTE-FORCE-SAMPLER which also generates unbiased size estimations, and HIDDEN-DB-SAMPLER which can be used with CAPTURE-&-RECAPTURE to produce biased size estimations.

**Comparison with BRUTE-FORCE-SAMPLER:** Both our random drill-down technique and BRUTE-FORCE-SAMPLER use a random process to select valid queries for estimating the size of a hidden database. The key difference is the success probability of such a random process. As we discussed in Section 2, BRUTE-FORCE-SAMPLER has an extremely low success rate because the database size is usually orders of magnitude smaller than the query space. Our approach, on the other hand, guarantees to find one (top-)valid query in every trial of the random walk, thanks to a combination of random drill-down and backtracking techniques. Such a difference on the success rate leads to a dramatic difference on the query cost. For example, consider a Boolean database with  $m$  tuples,  $n$  attributes, and  $k = 1$ . While BRUTE-FORCE-SAMPLER requires an average of  $2^n/m$  queries to find a valid query and produce an estimate, we require at most  $n$  queries (i.e., the longest possible drill-down) to do so.

**Comparison with HIDDEN-DB-SAMPLER:** There are two key differences between the random drill-down process in our approach and that in HIDDEN-DB-SAMPLER [13]. The first is our introduction of *backtracking*. With HIDDEN-DB-SAMPLER, the random drill-down process incurs an “early termination” (i.e., restarts from the root) if it reaches an underflowing node. As a result, one has to know the probability of early termination  $p_E$  in order to compute the probability of reaching a query  $q$  with  $h$  predicates:

$$p(q) = \frac{1}{(1 - p_E) \cdot \prod_{i=1}^h |Dom(A_i)|} \quad (3)$$

Unfortunately, it is extremely difficult to approximate  $p_E$  to the degree that supports an accurate estimation of  $p(q)$ . The reason is that  $(1 - p_E) \approx 0$  for many hidden databases, especially categorical ones with large fan-out attributes, whose sizes are order of magnitude smaller than the domain of all possible values. As a result, an extremely large number of random drill-downs must be taken before  $p(q)$  can be accurately estimated - which makes HIDDEN-DB-SAMPLER impossible to use for estimating the database size. Our technique, on the other hand, introduces backtracking to ensure that no random drill-down terminates without reaching a valid query, thereby enabling the *precise computation* of  $p(q)$ . It is the introduction of backtracking that essentially enables us to produce an unbiased estimate for the database size.

The second difference is on the sampling technique: HIDDEN-DB-SAMPLER uses rejection sampling which discards results of short random drill-downs with high probability to approximate a uniform selection probability for all tuples. Our approach, on the other hand, uses a *weighted sampling* technique which accepts all results and associate each with its selection probability  $p(q)$ . Then, the final estimate is adjusted by  $p(q)$  to produce an unbiased estimation. This change makes our approach a highly efficient algorithm for many hidden databases, because a random drill-down process always produces an estimate. In comparison, HIDDEN-DB-SAMPLER may have the reject a large number of random drill-downs before accepting the one as a uniform random sample.

### 3.3.2 Disadvantages on Estimation Variance

While our backtracking-enabled drill-down technique produces unbiased estimates, note that the mean squared error of an estimator depends on not only bias but also variance. A disadvantage of our approach is that it may incur high estimation variance for a database with highly skewed distribution, as illustrated by the following theorem. Recall that  $\Omega_{TV}$  is defined as the set of all *top-valid nodes* i.e., valid nodes that have overflowing parents.

**THEOREM 2. [Estimation Variance]** *The estimation generated by the random drill-down process over a categorical database has*

variance

$$s^2 = \left( \sum_{q \in \Omega_{TV}} \frac{|q|^2}{p(q)} \right) - m^2. \quad (4)$$

The proof directly follows from the variance definition.

Observe from Theorem 2 that the estimation variance can be large when there are deep top-valid queries in the tree, which usually occur in database with skewed distribution. In particular, consider a Boolean database with  $n + 1$  tuples  $t_0, t_1, \dots, t_n$  that satisfy the following condition: For any  $t_i$  ( $i \in [1, n]$ ),  $t_i$  has the opposite values as  $t_0$  on attributes  $a_{n-i+1}, \dots, a_n$ , and the same values on all other attributes i.e.,  $\forall j \in [1, n - i], t_0[a_j] = t_i[a_j]$ ,  $\forall j \in [n - i + 1, n], t_0[a_j] = 1 - t_i[a_j]$ . Figure 4 illustrates this scenario when  $k = 1$ . Note that all underflowing branches are omitted in the figure.

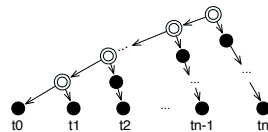


Figure 4: A Worst-Case Scenario

One can see that, when  $k = 1$ , there are two  $n$ -predicate queries (e.g.,  $t_0$  and  $t_1$ ) in the tree that both are top-valid queries. Note that the value of  $p(q)$  for these two queries are  $1/2^n$ . Thus, the variance is at least  $s^2 > 2^{n+1} - m^2$ . Since the domain size is usually order of magnitude larger than the database size i.e.,  $2^n \gg m$ , the variance can remain extremely large even after a substantial number of random drill-downs. For example, if the database has 40-attribute and one million tuples, at least  $10^{12}$  random drill-downs are required to reduce the variance to  $m^2$ , which still leads to significant error. The following corollary illustrates the generic case for categorical databases with an arbitrary  $k$ .

**COROLLARY 1. [Worst-Case Estimate Variance for BOOL-UNBIASED-SIZE]** *The worst-case estimate variance for an  $n$ -attribute,  $m$ -tuple hidden database with a top- $k$  interface satisfies*

$$s^2 > k^2 \cdot \left( \prod_{i=1}^{n-1} |Dom(A_i)| \right) - m^2. \quad (5)$$

Due to space limitations, we do not include the proof of the corollary. Note that this corollary shows a probabilistic lower bound which still assumes the randomness of the drill-down process (i.e., it considers the worst-case database instance, not the worst-case random walk). One can see from the corollary that the variance may be even more significant in the generic case, because the variance increases linearly with  $k^2$ . The next section is an effort dedicated to reduce the estimation variance.

## 4. VARIANCE REDUCTION

In this section, we describe two ideas to reduce the estimation variance: *weight adjustment* and *divide-&-conquer*.

### 4.1 Weight Adjustment

Weight adjustment is a popular technique for variance reduction in sampling traditional databases. In this subsection, we describe how weight adjustment can be applied over hidden databases<sup>3</sup>. In particular, we show that while weight reduction in general reduces estimation variance, the remaining variance is still significant for certain hidden databases with highly skewed distributions.

<sup>3</sup>Note that while the COUNT-based sampling part of the ALERT-HYBRID algorithm [14] can also be considered weighted sampling, the key differ-

### 4.1.1 Main Idea

The random drill-down process in BOOL-UNBIASED-SIZE essentially performs a sampling process over all top-valid nodes of the hidden database, with varying selection probability for different nodes. As any weighted-sampling estimator, the estimation variance is determined by the alignment between the selection probability distribution and the distribution of the measure attribute (i.e., the attribute to be aggregated). For our purpose, the measure aggregate is  $|q|$ , the size of a top-valid node. With BOOL-UNBIASED-SIZE, the selection probability for  $q$  is  $p(q)$ , which is generally independent of its measure attribute  $|q|$ . Thus, in some cases, the selection probability may be perfectly aligned with the measure attribute. For example, when  $k = 1$  and all top-valid nodes have exactly  $\log_2 m$  predicates, we have  $p(q) = 1/m$  and 0 variance according to Theorem 2. Nonetheless, there are also cases where these two distributions are badly misaligned which lead to an extremely large variance. An example of such a case was depicted in Figure 4 - where certain top-valid nodes reside on the leaf level, much deeper than other top-valid nodes.

The main idea of *weight adjustment* is to adjust the probability for following each branch (and thereby change the eventual selection probability for top-valid nodes), so as to better align the selection probability distribution  $p(q)$  with the distribution of  $|q|$ . The ideal objective is to achieve the perfect alignment with 0 variance - i.e., each top-valid node  $q$  has  $p(q) = |q|/m$ . Of course, such a perfect alignment is not possible without complete knowledge of the measure attribute distribution. However, prior “pilot” samples can help in estimating this distribution, as we shall discuss as follows.

To understand how to approximate this perfect alignment from the pilot samples, consider a node  $q_P$  with  $w$  branches  $q_{C_1}, \dots, q_{C_w}$ . Each branch  $q_{C_i}$  defines a subset of the database  $D_{C_i}$  consisting of all tuples that have corresponding top-valid nodes “under”  $q_{C_i}$ . That is, all tuples in  $D_{C_i}$  match the selection conditions of  $q_{C_i}$ . For example, in Figure 1,  $t_1, \dots, t_4$  are under  $q'_1$  while  $t_5$  and  $t_6$  are under  $q_1$ . The optimal alignment is to select each branch  $q_{C_i}$  with probability proportional to the size of its corresponding sub-database i.e.,  $|D_{C_i}|$ . For example, in the running example depicted in Figure 1,  $q'_1$  and  $q_1$  should be chosen with probability  $4/6$  and  $2/6$ , respectively.

If the precise size of  $D_{C_i}$  is known for every branch, then the probability for each top-valid node  $q$  to be picked up is  $p(q) = |q|/m$ , which leads to a perfect alignment that produces 0 variance. Without knowledge of  $|D_{C_i}|$ , we estimate it from the prior samples of top-valid nodes. In particular, let  $q_{H_1}, \dots, q_{H_s}$  be the historic top-valid queries reached under  $q_{C_i}$ , we estimate  $|D_{C_i}|$  as

$$|D_{C_i}| \approx \frac{1}{s} \cdot \sum_{j=1}^s \frac{|q_{H_j}|}{p(q_{H_j}|q_{i_j})}, \quad (6)$$

where  $p(q_{H_j}|q_{i_j})$  is the conditional probability for the random drill-down process to reach  $q_{H_j}$  given the fact that it reaches  $q_{C_i}$ . Note that  $p(q_{H_j}|q_{i_j})$  can be computed based on branch selection probability for each branch on the path from  $q_{i_j}$  to  $q_{C_i}$ . Consider the running example in Figure 1. If there is one historic drill down (without weight adjustment) through  $q_1$  which hits  $q_4$ , then we estimate the subtree size for  $q_1$  as  $1 \cdot (1/2)/(1/4) = 2$ , where  $1/2$  and  $1/4$  are the probability for the random walk to reach  $q_1$  and

ence between ALERT-HYBRID and the technique discussed here is how the weights are computed are the implications of imperfectly estimated weights. In ALERT-HYBRID, the weights are estimated based on a small pool of pilot sample tuples, and estimation errors on weights lead to biased samples [14]. Here, the weights are determined by COUNT estimations from prior drill downs, and imperfectly estimated weights do not affect the unbiasedness of BOOL-UNBIASED-SIZE.

$q_4$ , respectively. Intuitively,  $p(q_{H_j})/p(q_{i_j})$  is the conditional probability of reaching  $q_{H_j}$  had the random walk started at node  $q_{i_j}$ . Thus, the estimation in (6) can be derived in analogy to our size estimation in BOOL-UNBIASED-SIZE.

**Unbiasedness:** Note that weight adjustment does *not* affect the unbiasedness no matter how accurate the estimation of  $|D_{C_i}|$  is. The reason is that we always know precisely the probability we used to follow each branch, and therefore can compute  $p(q)$  precisely for each top-valid node reached by a random drill-down.

### 4.1.2 Effectiveness on Variance Reduction

The power of weighted sampling (a.k.a. importance sampling) on reducing estimation variance has been well studied in statistics (e.g., [26]) and database sampling [4]. Unfortunately, for the purpose of this paper, weight adjustment still cannot address the worst-case scenario depicted in Figure 4, where the existence of deep top-valid nodes leads to an extremely large estimation variance. The difficulty for weight adjustment to address this scenario comes from two perspectives: First, such a deep-level node is unlikely to be reached by a historic drill-down, leading to low probability of applying the weight adjustment in the first place. Second, even with historic trials hitting the node, the relative error is likely to be high for estimating the size of a small subtree.

**COROLLARY 2. [Worst-Case Estimate Variance with Weight Adjustment]** *For an  $n$ -attribute,  $m$ -tuple hidden database, after  $r$  random drill-downs, the worst-case estimation variance generated by the random-drill down with weight adjustment satisfies*

$$s^2 \geq \frac{2^{n-\log_2 r} \cdot m}{n - \log_2 r + 1} - m^2. \quad (7)$$

The corollary shows the worst-case scenario still generates unacceptably high variance even after applying weight adjustment. Note that it is again a probabilistic lower bound which assumes the randomness of the drill-down process. According to Corollary 2, for a 40-attribute 100,000-tuple database,  $s^2 \geq 354.29 \cdot m^2$  even after 1,000 random drill-downs have been performed.

## 4.2 Divide-&-Conquer

We now describe the idea for divide-&-conquer, a variance reduction technique which is independent of weight adjustment but can be used in combination with it. As mentioned in the introduction, divide-&-conquer provides the most significant variance reduction especially for the worst-case scenarios.

### 4.2.1 Motivation and Technical Challenge

Our discussions for Corollary 1 and 2 indicate *deep top-valid nodes* as the main source of high variance before and after weight adjustment, respectively. More fundamentally, the cause for the existence of deep top-valid queries lies on the significant difference between the database size  $m$  and the domain size  $|Dom|$ . To understand why, consider the case where  $k = 1$ . A deepest (i.e., leaf-level) top-valid query returns  $|Dom|$  as the estimation, while the actual size is  $m$ . The following theorem further formalizes such a fundamental cause of high variance:

**THEOREM 3. [Upper Bound on Estimation Variance]** *When  $k = 1$ , the estimation variance of a random drill down satisfies*

$$s^2 \leq m^2 \cdot \left( \frac{|Dom|}{m} - 1 \right). \quad (8)$$

The theorem shows a large value of  $|Dom|/m$  as a cause of high estimation variance of HD-UNBIASED. The main motivation for

divide-&-conquer is to partition the originally large domain into smaller, mutually exclusive, subdomains so as to reduce  $|Dom|$  for the subdomains. Note that the partitioning strategy must be carefully designed so  $m$  will not be significantly reduced. To understand why, consider an otherwise simple strategy of dividing the domain randomly into  $b$  mutually exclusive subdomains, using HD-UNBIASED-SIZE to estimate each, and take the sum as the total size estimation. This partitioning strategy reduces  $|Dom|$  and  $m$  with the same ratio  $b$ , leaving  $|Dom|/m$  unchanged. As a result, according to Theorem 3, the total estimation variance is only reduced by a factor of  $b$ , which is offset by the additional queries required for executing HD-UNBIASED-SIZE over the  $b$  subdomains.

One can see that, to solve this problem, the domain must be partitioned in a way such that while the number of subdomains may be large (such that each subdomain can be small), the vast majority of tuples appear in only a small number of subdomains. Then, for these subdomains, the ratio of subdomain size over the number of tuples in the subdomain is small, allowing a reduced estimation variance for the total database size. Note that the other sparsely packed subdomains will not adversely affect the estimation variance to a significant degree because of the limited number of tuples contained in the subdomains.

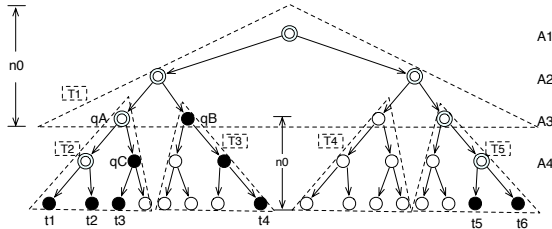


Figure 5: An Example of Tree-based Partitioning

#### 4.2.2 Main Idea

To gather most tuples into a few small subdomains, we propose a query-tree-based recursive partitioning of the domain. We start with Boolean databases and then extend the results to categorical data at the end of this subsection. Consider the tree shown in Figure 1 which is also depicted in Figure 5. Originally, the domain includes the entire tree which includes a top-valid node for each tuple. We partition the domain into subdomains corresponding to subtrees with  $n_0$  levels. Figure 5 depicts an example of such a partition with  $n_0 = 3$ . Consider the root to be the first level. The partition generates one subtree from the first to the  $n_0$ -th level,  $2^{n_0-1}$  subtrees from the  $n_0$ -th to the  $(2n_0 - 1)$ -th level, etc. For example,  $T_1$  in Figure 5 is the 1-to-3 level subtree, and  $T_2, \dots, T_5$  are the four 3-to-5 level subtrees. Note that the root node of a  $(i \cdot n_0 - i + 1)$ -to- $(i \cdot n_0 - i + n_0)$ -th level subtree is also a bottom-level node of a  $(i \cdot n_0 - 2i + 1)$ -to- $(i \cdot n_0 - 2i + n_0)$ -th level subtree. For example, the root node of  $T_2$  is a leaf-level node of  $T_1$  in Figure 5. A tuple is included in a subtree iff its corresponding top-valid node is a non-root node of the subtree. For example, in Figure 5,  $t_1$  and  $t_4$  are included in  $T_2$  and  $T_3$ , respectively. One can see that each tuple belongs to one and only one subtree. Thus, the subtrees form a mutually exclusive partition of the original domain.

With the subtree partitioning strategy, our size estimation algorithm can be stated as the following recursive procedure: We start with the top-most subtree (e.g.,  $T_1$  in Figure 5) and start to perform the random drill-down process over the subtree. In particular, we perform  $r$  random drill-downs where  $r$  is a pre-determined parameter, the setting of which will be discussed in Section 5. Note that each random drill-down may terminate at two types of nodes: a

top-valid node or a bottom-level overflowing node of the subtree, which we refer to as a *bottom-overflow* node.

For each top-valid node  $q_{TV}$  which terminates a random drill-down of the top-most subtree, we compute  $\delta(q_{TV}) = r \cdot p(q_{TV})$ , where  $p(q_{TV})$ , as defined in Section 3, is the probability for a random drill-down to reach  $q_{TV}$ . Intuitively,  $r \cdot p(q_{TV})$  is the expected number of random drill-downs that will terminate at  $q_{TV}$ . For example, in Figure 5, with  $r = 2$  random drill-downs per subtree, we have  $\delta(q_B) = 2 \cdot (1/4) = 1/2$ . Note that when  $r = 1$ ,  $\delta(q_{TV}) = p(q_{TV})$ .

For each bottom-overflow node  $q_{BO}$  which terminates a random drill-down of the top-most subtree, we perform the random drill-down process over the subtree with root of  $q_{BO}$ , again with  $r$  drill-downs. This recursive process continues until no bottom-overflow node is discovered. For a top-valid node  $q'_{TV}$  captured in a subtree with root of  $q_R$ , we compute

$$\delta(q'_{TV}) = r \cdot p(q'_{TV}) \cdot \delta(q_R), \quad (9)$$

where  $p(q'_{TV})$  is the probability for a random drill-down over the subtree (of  $q_R$ ) to reach  $q'_{TV}$ , and  $\delta(\cdot)$  for a bottom-overflow node  $q_R$  is defined in the same way as for a top-valid node in (9). Again,  $\delta(q'_{TV})$  is intuitively the expected number of drill-downs that terminate at  $q'_{TV}$ . For example, in Figure 5 with  $r = 2$ , we have  $\delta(q_A) = 1/2$  and  $\delta(q_C) = 2 \cdot (1/2) \cdot (1/2) = 1/2$ . Note that when  $r = 1$ , there is  $\delta(q) = p(q)$  for all nodes in all subtrees, essentially reverting the random drill-down process to the state without divide-&-conquer.

Let  $Q_{TV}$  be the set of top-valid nodes captured by the random drill-downs over all subtrees. We estimate the database size as

$$\tilde{m} = \sum_{q \in Q_{TV}} \frac{|q|}{\delta(q)}. \quad (10)$$

The unbiasedness of this estimator follows from Theorem 1.

For categorical data, the only change required is the depth of each subtree. Instead of having  $n_0$  levels for all subtrees, we maintain a (approximately) constant domain size for each subtree. As a result, a categorical subtree could be shallow or deep depending on the fan-outs of attributes involved. In particular, we set  $D_{UB}$  as an upper bound on the domain size ( $D_{UB} \geq \max_i |Dom(A_i)|$ ). Each subtree should have the maximum number of levels without exceeding  $D_{UB}$ . In the running example, if  $A_1, \dots, A_5$  is the attribute order and  $D_{UB} = 10$ , then Levels 1-4 (i.e.,  $A_1, A_2, A_3$  with domain size of  $2^3 = 8$ ) and 4-6 (i.e.,  $A_4, A_5$  with domain size of  $2 \times 5 = 10$ ) become the two layers of subtrees. One can see that the unbiasedness is unaffected by the change.

#### 4.2.3 Effectiveness on Variance Reduction

Divide-&-conquer is effective on reducing the estimation variance because it provides a significantly better alignment between the selection probability distribution for top-valid nodes and the measure attribute distribution. To understand why, consider a Boolean database with  $k = 1$  and two top-valid nodes  $q$  and  $q'$ , at the second level (i.e., as a child of the root) and the bottom-level (i.e.,  $n + 1$ -th level), respectively. Without divide-&-conquer, at the first drill-down,  $q$  has selection probability of  $1/2$  while  $q'$  may have selection probability as small as  $p(q') = 1/2^n$ . This forms a striking contrast with the uniform distribution of the measure attribute (i.e.,  $|q|/m = |q'|/m = 1/m$  for each top-valid node), leading to a bad alignment between the two.

With divide-&-conquer, the selection probability for a deep top-valid node like  $q'$  is significantly increased, while that for a shallow top-valid node like  $q$  remains the same. In particular, the expected number of random drill-downs that choose  $q$  is  $\delta(q) = r/2$ . The

expected number for  $q'$  is at least

$$\delta(q') \geq \frac{r^{\lfloor \frac{n-1}{n_0-1} \rfloor}}{2^n}, \quad (11)$$

where  $n_0$  is the depth of a subtree and  $r$  is the number of drill downs conducted over each subtree. One can see that the difference between  $\delta(q)$  and  $\delta(q')$  is reduced by a factor of  $r^{\lfloor (n-1)/(n_0-1) \rfloor}$  after divide-&-conquer, leading to a better alignment with the measure attribute distribution.

The total number of queries issued by the divide-&-conquer technique depends on the underlying data distribution. While theoretically a large number of queries may be issued, in practice the query cost is usually very small due to two reasons: (1) One can see from (11) that even a very small  $r$  can significantly improve the alignment and thereby reduce the estimation variance. (2) As the experimental results show, for real-world hidden databases, even with a highly skewed distribution, the top-valid nodes are likely to reside on a small number of subtrees. Furthermore, the following theorem shows that with the same query cost divide-&-conquer can significantly reduce the worst-case estimation variance.

**THEOREM 4. [Estimation Variance with D&C]** *When  $n$  is sufficiently large, for a given number of queries, in the worst case scenario where a random drill-down without divide-&-conquer generates the largest variance  $s^2$ , the estimation variance with D&C,  $s_{DC}^2$ , satisfies*

$$\frac{s^2}{s_{DC}^2} = O\left(\frac{r^{\log_{D_{UB}} |Dom(A_1, \dots, A_n)|}}{\log_{D_{UB}} |Dom(A_1, \dots, A_n)|}\right), \quad (12)$$

where  $r$  is the number of drill-downs performed for each subtree, and  $D_{UB}$  is an upper bound on the subdomain size of a subtree.

The proof of the theorem is not included due to space limitation.

## 5. DISCUSSIONS OF ALGORITHMS

### 5.1 HD-UNBIASED-SIZE

**Parameter Settings:** By combining the three ideas, i.e., backtracking-enabled random walk, weight adjustment and divide-&-conquer, Algorithms HD-UNBIASED-SIZE features two parameters:  $r$ , the number of random drill-downs performed on each subtree, and  $D_{UB}$ , the upper bound on the domain size of a subtree. While neither parameter affects the unbiasedness of estimation, they both affect the estimation variance as well as query efficiency. At the extreme cases, divide-&-conquer is disabled when  $r = 1$  or  $D_{UB} = |Dom(A_1, \dots, A_n)|$ , and crawling essentially occurs when  $r$  is extremely large or  $D_{UB}$  is extremely small. Thus, an appropriate setting for  $r$  and  $D_{UB}$  should make a tradeoff between estimation variance and query cost. Theoretical derivation of the optimal values for  $r$  and  $D_{UB}$  is difficult because of their dependencies on the underlying data distribution. Fortunately, as indicated by Theorem 4 and verified by our experiments, the depth of a subtree is the most important factor that determines the estimation variance because the variance changes exponentially with the depth.  $r$ , on the other hand, is not a very sensitive factor for the estimation variance. Thus, to perform HD-UNBIASED-SIZE over a hidden database, one should first determine  $D_{UB}$  according to the variance estimation. Then, starting from  $r = 2$ , one can gradually increase the budget  $r$  until reaching the limit on the number of queries issuable to the hidden database.

**Attribute Order:** We propose to arrange the attributes in decreasing order of their fanouts (i.e.,  $|Dom(A_i)|$ ) from the root to the

leaf level of the query tree. The reason lies on the query cost with smart backtracking. Recall from Section 3.2 that, with smart tracking, the expected number of branches one need to test for a given node is  $QC = 1 + \sum_{j=1}^w (w_U(j) + 1)^2/w$ , where  $w_U(j)$  is the number of underflowing branches immediately preceding the  $j$ -th branch. One can see that  $w_U(j)$  is in general minimized when a high fanout attribute is placed at the top levels of the tree. Thus, the overall query cost will be reduced by sorting all attributes from largest to smallest domains.

### 5.2 HD-UNBIASED-AGG

In this section we discuss HD-UNBIASED-AGG, by extending HD-UNBIASED-SIZE to answer SUM and AVG aggregate queries, as well as queries with selection conditions. While our sampler provides unbiased estimates of SUM queries, we point out that it cannot provide unbiased estimations of AVG queries.

**For answering SUM and AVG queries:** The same random drill-down process can be used to generate unbiased estimate for SUM queries. There is only a slight change on computing the estimation. Consider a SUM query `SELECT SUM( $A_i$ ) FROM  $D$` . For a random drill-down process (with divide-&-conquer), an unbiased estimator for the SUM query is

$$\sum_{q \in Q_{TV}} \sum_{t \in q} \frac{t[A_i]}{\delta(q)}, \quad (13)$$

where  $\sum_{t \in q} t[A_i]$  is the sum of attribute values  $A_i$  for all tuples in  $q$ , and  $Q_{TV}$  is the set of top-valid nodes captured by the random drill-downs over all subtrees.

However, note that the random drill-down process cannot be used as an unbiased estimator for AVG queries. The direct division of the unbiased SUM and COUNT estimators lead to a biased AVG estimator. This is consistent with the observation in [13] that it is extremely difficult to generate unbiased AVG estimations without issuing a very large number of queries (e.g., by using BRUTE-FORCE-SAMPLER).

**For answering queries with selection conditions:** Our previous discussions have been focused on queries that select all tuples in the database. The random drill-down approach can also generate unbiased SUM and COUNT estimates for queries with conjunctive selection conditions. In particular, a conjunctive query can be considered as selecting a subtree which is defined with a subset of attributes (as levels) and, for each attribute involved, a subset of its values (as branches). The random drill-down approach can be applied to the subtree directly to generate unbiased estimations.

## 6. EXPERIMENTS AND RESULTS

In this section, we describe our experimental setup and present the experimental results. We carry out empirical studies to demonstrate the superiority of HD-UNBIASED-SIZE and HD-UNBIASED-AGG over BRUTE-FORCE-SAMPLER and CAPTURE-&-RECAPTURE discussed in Section 2. We also draw conclusions on the individual impact of weight adjustment and divide-&-conquer on reducing the estimation variance.

### 6.1 Experimental Setup

1) *Hardware and Platform:* All our experiments were performed on a 1.99 Ghz Intel Xeon machine with 4 GB of RAM. The HD-UNBIASED-SIZE and HD-UNBIASED-AGG algorithms was implemented in MATLAB for testing offline datasets, and PHP for executions over the real Yahoo! Auto website.

2) *Data Sets:* Recall that we proposed three algorithms in the paper: BOOL-UNBIASED-SIZE, which applies only to Boolean

data, and HD-UNBIASED-SIZE/AQP which apply to both Boolean and categorical data. To test their performance, we consider both Boolean and categorical datasets. To properly evaluate the accuracy of the estimations, we need to know the ground truth on the aggregates being estimated. Thus, we perform the performance comparison experiments on offline datasets to which we have full access. Meanwhile, to demonstrate the practical impact of our techniques, we also test the algorithms over an online hidden database. Both offline and online databases are described as follows. For all databases, we set  $k = 100$  unless otherwise specified.

**Boolean Synthetic:** We generated two Boolean datasets, each of which has 200,000 tuples and 40 attributes. The first dataset is generated as i.i.d. data with each attribute having probability of  $p = 0.5$  to be 1. We refer to this dataset as the *Bool-*iid** dataset. The second dataset is generated in a way such that different attributes have different distribution. In particular, there are 40 independent attributes. 5 have probability of  $p = 0.5$  to be 1, while the others have the probability of 1 ranging from  $1/70$  to  $35/70$  with step of  $1/70$ . One can see that this dataset features a skewed distribution. We refer to it as the *Boolean-mixed* dataset.

**Offline Yahoo! Auto:** The offline Yahoo! Auto dataset consists of data crawled from the Yahoo! Auto website <http://autos.yahoo.com/>, a real-world hidden database, in 2007. In particular, the original crawled dataset contains 15,211 used cars for sale in the Dallas-Fort Worth metropolitan area. We enlarged the dataset to 188,790 tuples by following the original distribution of the small dataset, in order to better test the ability of our algorithms over large databases. In particular, the DBGen synthetic data generator [16] was used. There are a total of 38 attributes including 32 Boolean ones, such as A/C, POWER LOCKS, etc, and 6 categorical attributes, such as MAKE, MODEL, COLOR, etc. The domain sizes of categorical attributes range from 5 to 16.

**Online Yahoo! Auto:** We also tested our algorithms over the real-world web interface of Yahoo! Auto, a popular hidden database. In particular, we issue queries through the advanced used car search interface available at [http://autos.yahoo.com/listings/advanced\\_search](http://autos.yahoo.com/listings/advanced_search). A specific requirement of this webpage is that either MAKE/MODEL or ZIP CODE must be specified for a query to be processed. To address this requirement, we place the MAKE/MODEL attribute at the top of our query tree and issue every query with its value specified.

3) *Aggregate Estimation Algorithms:* We tested three algorithms in this paper: BOOL-UNBIASED-SIZE, HD-UNBIASED-SIZE, and HD-UNBIASED-AGG. BOOL-UNBIASED-SIZE is parameterless, while both HD-UNBIASED-SIZE and HD-UNBIASED-AGG feature two parameters:  $r$ , the number of drill-downs performed over each subtree, and  $D_{UB}$ , the maximum subdomain size for each subtree. We tested our algorithms with various parameter settings to illustrate how  $r$  and  $D_{UB}$  can be properly set in practice.

We also tested two baseline aggregate-estimation algorithms discussed in Section 2: BRUTE-FORCE-SAMPLER [13] and the use of CAPTURE-&-RECAPTURE with HIDDEN-DB-SAMPLER [13]. BRUTE-FORCE-SAMPLER cannot return any result during our test of issuing 100,000 queries, because of the drastic difference between the size of the database and set of all possible tuples. Thus, we compared the performance of our algorithms with CAPTURE-&-RECAPTURE in the experimental results.

4) *Performance Measures:* For query cost, we focus on the number of queries issued through the web interface of the hidden database. For estimation accuracy, we tested three measures: (1) the mean squared error (MSE), (2) the relative error (i.e.,  $|\tilde{\theta} - \theta|/\theta$  for an estimator  $\tilde{\theta}$  of aggregate  $\theta$ ), and (3) error bars (indicating one standard deviation of uncertainty).

## 6.2 Experimental Results

We compared the performance of HD-UNBIASED-SIZE, BOOL-UNBIASED-SIZE and CAPTURE-&-RECAPTURE over a Boolean database. For HD-UNBIASED-SIZE, we set parameters  $r = 4$  and  $D_{UB} = 2^5$ . Figure 6 depicts the tradeoff between MSE and query cost for the three algorithms for BOOLEAN-*iid* and BOOLEAN-*mixed* datasets. One can see from the figure that for both datasets, BOOL-UNBIASED-SIZE and HD-UNBIASED-SIZE generates orders of magnitude smaller MSE than CAPTURE-&-RECAPTURE. Compared with BOOL-UNBIASED-SIZE, HD-UNBIASED-SIZE is capable of further reducing the MSE by up to an order of magnitude (for BOOLEAN-MIXED) thanks to the integration of weight adjustment and divide-&-conquer, the individual effect of each will be discussed later in this section.

Another observation from Figure 6 is that the MSE for Boolean-*mixed* is higher than that for Boolean-*iid*. This is consistent with our discussions in Section 3.3.2 which show that the MSE is higher over a skewed data distribution, like that of BOOLEAN-Mixed.

To provide an intuitive demonstration of the estimation accuracy, Figure 7 depicts the tradeoff between relative error and query cost for the three algorithms under the same settings as Figure 6. Figure 8 further shows the error bars for HD-UNBIASED-SIZE. One can see that both BOOL-UNBIASED-SIZE and HD-UNBIASED-SIZE are capable of producing smaller than 2% relative error for both datasets with fewer than 500 queries. In particular, all error bars of HD-UNBIASED-SIZE are within the range of 99%-101.5%. This shows that our algorithm is capable of producing accurate estimates even for a database with skewed underlying distribution, like Boolean-Mixed. Figures 9 and 10 depict the performance of HD-UNBIASED-SUM over the SUM of a randomly chosen attribute. The observations are similar to the COUNT case.

We tested HD-UNBIASED-SIZE with varying database size  $m$ . Figures 11 and 12 depict the change of MSE and query cost, respectively, when  $m$  varies from 50,000 to 300,000. The parameters are  $r = 4$  and  $D_{UB} = 16$ . One can see from the figure that the MSE increases (approximately) linearly with the database size. This is consistent with our theoretical analysis in Theorem 3. The query cost also increases linearly with  $m$ , showing the scalability of our algorithm to large databases. One can observe from the figures that while the increase of query cost with  $m$  is always equal for Boolean-*iid* and Boolean-Mixed, the difference between their MSE become larger when  $m$  increases. This is because the larger the Boolean-Mixed database is, the “more skewed” its distribution on the query tree will be, leading to a larger estimation variance.

To study how the value of  $k$  for the top- $k$  interface affects the performance of our algorithm, we tested HD-UNBIASED-SIZE with  $k$  ranging from 100 to 500. The changes of MSE and query cost are shown in Figure 13. One can see that from the figure that with a larger  $k$ , both MSE and query cost decreases, leading to a more efficient and accurate estimation.

We also studied the individual effects of weight adjustment and divide-&-conquer to variance reduction. In particular, we tested the performance of the following four algorithms over the categorical offline Yahoo! Auto dataset: (1) HD-UNBIASED-SIZE, (2) HD-UNBIASED-SIZE without weight adjustment, (3) HD-UNBIASED-SIZE without divide-&-conquer (i.e., by setting  $r = 1$ ), and (4) HD-UNBIASED-SIZE with neither weight adjustment nor divide-&-conquer. For HD-UNBIASED-SIZE, we set  $r = 5$  and  $D_{UB} = 16$ . Figure 14 depicts the tradeoff between MSE and query cost for all four algorithms. Figure 15 further shows the error bars for (1) which has the best performance.

We studied how the parameters of HD-UNBIASED-SIZE affect its performance. Figure 16 depicts the change of MSE and query

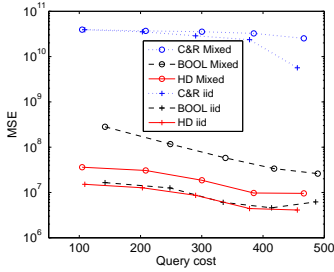


Figure 6: MSE vs. query cost

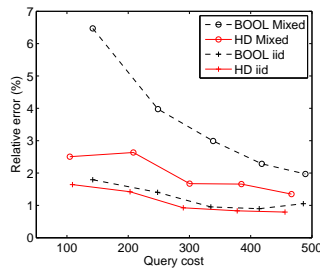


Figure 7: Relative error

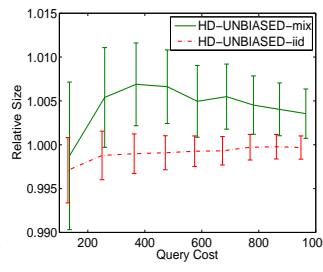


Figure 8: Error Bars

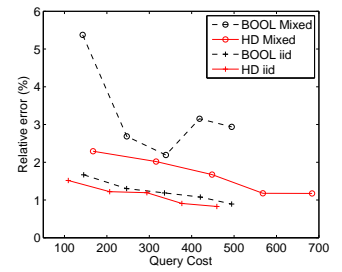


Figure 9: SUM relative error

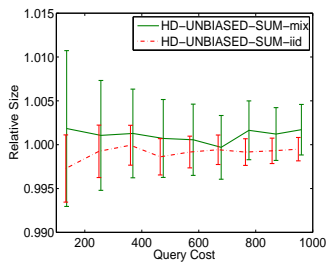


Figure 10: SUM error bars

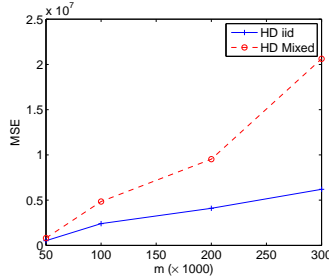


Figure 11: MSE vs.  $m$

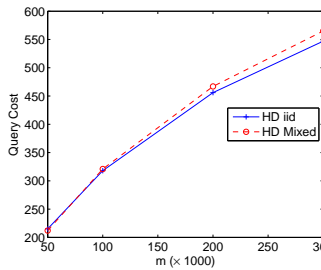


Figure 12: Query cost vs.  $m$

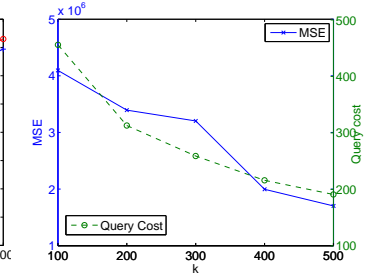


Figure 13: MSE, query cost vs.  $k$

cost when  $r$ , the number of drill-downs per subtree, varies between 4 and 8. We conducted the experiment with  $D_{UB} = 16$ . One can see that the larger  $r$  is, the more queries will be issued, and the smaller the estimation variance will be. This is consistent with our intuitive discussions in Section 5.1.

Figure 17 shows the change of MSE and query cost when  $D_{UB}$  varies between 16 and 104544 (the domain size of the database). The experiment was conducted when  $r = 5$ . One can see from the figure that the larger  $D_{UB}$  is, the fewer queries need to be issued, but the MSE will increase correspondingly.

We also tested the impact of  $r$  on the tradeoff between MSE and query cost. In particular, we set  $D_{UB} = 16$ . For each value of  $r$ , we repeat the execution of HD-UNBIASED-SIZE for certain number of times to reach a similar query cost. Then, we compute MSE based on the average of the estimations from the repeated runs. The results are as follows.

$r$	3	4	5	6	7	8
Query Cost	440	466	494	373	473	607
MSE ( $\times 10^{10}$ )	4.53	4.84	4.29	4.83	4.66	3.53

One can see that the tradeoff between MSE and query cost is not sensitive to the value of  $r$ . This verifies our discussions in Section 5.1 for the setting of  $r$ .

Finally, we tested HD-UNBIASED-SIZE and HD-UNBIASED-AGG over the real-world web interface of Yahoo! Auto. Note that Yahoo! enforces a limit on the frequency of queries issued from an IP address. This prevents us from issuing a large number of queries for the experiments. In particular, we conducted 10 executions of HD-UNBIASED-SIZE to estimate the number of Toyota Corollas in the database (issuing an average of 193 queries per execution). The parameters are set as  $r = 30$ ,  $D_{UB} = 126$ . Figure 18 shows the estimations generated after each round of execution. One can see from the figure that our estimations are close to 13613, the COUNT disclosed on the Yahoo! website. Figure 19 shows estimations generated by HD-UNBIASED-AGG for the total inventory balance (i.e., sum of prices) for cars of five popular models, with

up to 1,000 queries issued for each estimation. The ground truth on such information is not disclosed on the Yahoo! website.

## 7. RELATED WORK

**Crawling and Sampling from Hidden Databases:** There has been prior work on crawling as well as sampling hidden databases using their public search interfaces. Several papers have dealt with the problem of crawling and downloading information present in hidden text based databases [1, 8, 23]. [2, 20, 25] deal with extracting data from structured hidden databases. [11] and [24] use query based sampling methods to generate content summaries with relative and absolute word frequencies while [17, 18] uses two phase sampling method on text based interfaces. [10, 12] discuss top-k processing which considers sampling or distribution estimation over hidden sources. In [13, 14] techniques have been developed for random sampling from structured hidden databases leading to the HIDDEN-DB-SAMPLER algorithm. Techniques to thwart such sampling attempts have been developed in [15].

**Sampling and Size Estimation for Search Engine's Corpus:** The increase in popularity of search engines has motivated the research community to develop techniques to discover its contents. [21, 28] studied the estimation by capture-recapture method to identify the index size of a search engine. [7] employed Monte Carlo methods to generate a near-uniform sampling from the search engine's corpus, while taking into consideration the degrees of documents and cardinalities of queries. With approximate document degrees, techniques for measuring search engine metrics were proposed in [5]. Sampling online suggestion text databases were discussed in [6] to significantly improve the service quality of search engines and to study users' search patterns.

## 8. CONCLUSION

In this paper we have initiated an investigation of the unbiased estimation of the size and other aggregates over hidden web databases through its restrictive web interface. We proposed backtrack-enabled random walk schemes over the query space to produce unbiased es-

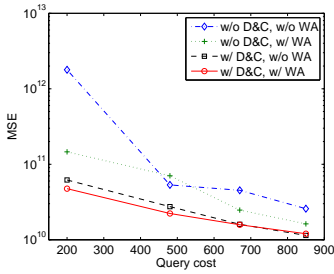


Figure 14: Individual effects

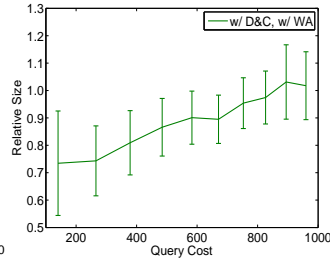


Figure 15: Yahoo! Auto

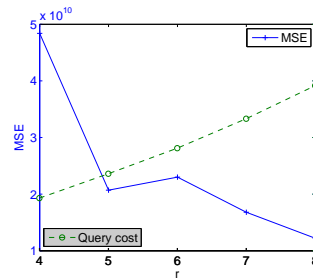


Figure 16: Effect of  $r$

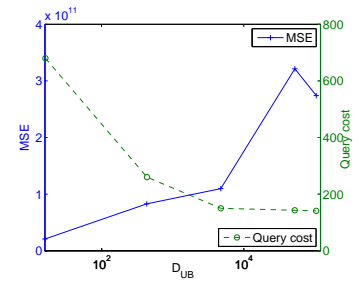


Figure 17: Effect of  $D_{UB}$

timates for SUM and COUNT queries, including the database size. We also proposed two ideas, weight adjustment and capture-&-recapture, to reduce the estimation variance. We provided theoretical analysis for estimation accuracy and query cost of the proposed ideas. We also described a comprehensive set of experiments that demonstrate the effectiveness of our approach over synthetic and real-world hidden databases.

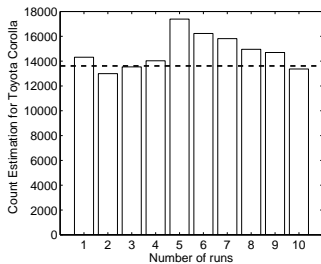


Figure 18: Toyota Corolla

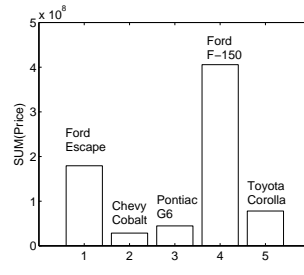


Figure 19: SUM(Price)

## 9. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their careful reading of the draft and thoughtful comments.

## 10. REFERENCES

- [1] E. Agichtein, P. G. Ipeirotis, and L. Gravano. Modeling query-based access to text databases. In *WebDB*, 2003.
- [2] M. Alvarez, J. Raposo, A. Pan, F. Cacheda, F. Bellas, and V. Carneiro. Crawling the content hidden behind web forms. In *ICCSA*, 2007.
- [3] S. C. Amstrup, B. F. J. Manly, and T. L. McDonald. *Handbook of capture-recapture analysis*. Princeton University Press, 2005.
- [4] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, 2003.
- [5] Z. Bar-Yossef and M. Gurevich. Efficient search engine measurements. In *WWW*, 2007.
- [6] Z. Bar-Yossef and M. Gurevich. Mining search engine query logs via suggestion sampling. In *VLDB*, 2008.
- [7] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine's corpus. *Journal of the ACM*, 55(5), 2008.
- [8] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. In *WWW*, 1998.
- [9] A. Broder, M. Fontura, V. Josifovski, R. Kumar, R. Motwani, S. U. Nabar, R. Panigrahy, A. Tomkins, and Y. Xu. Estimating corpus size via queries. In *CIKM*, 2006.
- [10] N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In *ICDE*, 2002.
- [11] J. P. Callan and M. E. Connell. Query-based sampling of text databases. *ACM TOIS*, 19(2):97–130, 2001.
- [12] K. C.-C. Chang and S. won Hwang. Minimal probing: supporting expensive predicates for top-k queries. In *SIGMOD*, 2002.
- [13] A. Dasgupta, G. Das, and H. Mannila. A random walk approach to sampling hidden databases. In *SIGMOD*, 2007.
- [14] A. Dasgupta, N. Zhang, and G. Das. Leveraging count information in sampling hidden databases. In *ICDE*, 2009.
- [15] A. Dasgupta, N. Zhang, G. Das, and S. Chaudhuri. Privacy preservation of aggregates in hidden databases: Why and how? In *SIGMOD*, 2009.
- [16] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger. Quickly generating billion-record synthetic databases. In *SIGMOD*, 1994.
- [17] Y.-L. Hedley, M. Younas, A. E. James, and M. Sanderson. A two-phase sampling technique for information extraction from hidden web databases. In *WIDM*, 2004.
- [18] Y.-L. Hedley, M. Younas, A. E. James, and M. Sanderson. Sampling, information extraction and summarisation of hidden web databases. *Data and Knowledge Engineering*, 59(2):213–230, 2006.
- [19] D. Horvitz and D. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47:663–685, 1952.
- [20] S. Little, D. Embley, D. Scott, and S. Yau. Extracting data behind web forms. In *ER (Workshops)*, 2002.
- [21] Y. C. Liu, K. and W. Meng. Discovering the representative of a search engine. In *CIKM*, 2002.
- [22] J. Lu. Efficient estimation of the size of text deep web data source. In *CIKM*, 2008.
- [23] A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden web content through keyword queries. In *JCDL*, 2005.
- [24] L. G. Panagiotis G. Ipeirotis. Distributed search over the hidden web: Hierarchical database sampling and selection. In *VLDB*, 2002.
- [25] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB*, 2001.
- [26] B. Ripley. *Stochastic Simulation*. Wiley & Sons, New York, 1987.
- [27] G. A. F. Seber. *The estimation of animal abundance and related parameters*. MacMillan Press, New York, 1982.
- [28] M. Shokouhi, J. Zobel, F. Scholer, and S. Tahaghoghi. Capturing collection size for distributed non-cooperative retrieval. In *SIGIR*, 2006.