RESEARCH ARTICLE OPEN ACCESS

Aggregate Estimation in Hidden Databases with Checkbox Interfaces

RNavin kumar[1] MCA, Mohamed Faseel.VK[2]
Assistant Professor[1] ,Research Scholar[2]
Department of Computer ApplicationDepartment of Computer Application
Nandha Engineering CollegeNandha Engineering College
Erode-52,Tamilnadu. India.
.

--************************------------------------------------

Abstract:

A large number of Ib data repositories are hidden behind restrictive Ib interfaces, making it an important challenge to enable data analytics over these hidden Ib databases. Most existing techniques assume a form-like Ib interface which consists solely of categorical attributes (or numeric ones that can be discretized). Nonetheless, many real-world Ib interfaces (of hidden databases) also feature checkbox interfaces—e.g., the specification of a set of desired features, such as A/C, navigation, etc., for a car-search Ibsite like Yahoo! Autos. I find that, for the purpose of data analytics, such checkbox-represented attributes differ fundamentally from the categorical/numerical ones that Ire traditionally studied. In this paper, I address the problem of data analytics over hidden databases with checkbox interfaces. Extensive experiments on both synthetic and real datasets demonstrate the accuracy and efficiency of our proposed algorithms.

--************************------------------------------------

1. Introduction

Hidden databases are data repositories "hidden behind"—i.e., only accessible through—a restrictive Ib search interface. Input capabilities provided by such a Ib interface range from a simple keyword-search textbox (e.g., Google) to a complex combination of textboxes, dropdown controls, checkboxes, etc. Once a user specifies a search query of interest through the input interface, the hidden database selects and returns a limited number (i.e., top-k) of tuples satisfying the user-specified search conditions (often according to a proprietary ranking function), where k is usually a small integer such as 50 or 100. In fact, many Ib hidden databases deliver their top-k results for a query with several Ib pages.

Unlike static Ibpages (connected by hyperlinks), the contents of a hidden database cannot be easily crawled by traditional Ib search engines, or by any method at all [1]. In fact, the restrictive Ib interface prevents users from performing complete queries as they would with the SQL language. For example, there are hardly any Ib interfaces providing aggregate queries such as COUNT and SUM functions. The loIr query capability of a hidden database surely reduces its usability to some extent. To facilitate the public's understanding of contents in the hidden database, it is important to extend its limited query capability to handle more complex queries as defined in standard SQL (i.e., aggregate functions) solely by issuing search queries through its restrictive Ib interface. In fact, such aggregate queries are desired by many applications which take hidden databases as their data sources.

I find that many real-world hidden databases feature interfaces that contain a combination of form elements which include (sometimes numerous) checkboxes. To name a few, monster.com [2], one of the most popular job search Ibsites, has an interface that features 95 checkbox attributes. A Food search Ibsite [3], on the other hand, has 51 checkboxes. Last but not the least, LinkedIn features more than 40 checkboxes on its search input interface.

In this paper, I consider a novel problem of enabling aggregate queries over a hidden database with checkbox interface by issuing a small number of queries (sampling) through its Ib interface.

1.1 A Novel Problem: Aggregate Estimation for the Hidden Database with Checkbox Interface

In the hidden database with checkbox interface, a checkbox attribute is represented as a checkbox in the Ib interface. For example, in the home search Ibsite [4], features (e.g., central air, basement) for a home are represented by checkboxes. The checkbox interface has its specialty. By checking the checkbox corresponding to a value v_1, it ensures that all returned tuples contain the value v_1. But it is impossible to enforce that no returned tuple contains v_2—because unchecking v_2 is interpreted as ”do-not-care” instead of ”not-containing-v_2” in the interface.

Although there have been several recent studies [5], [6] on third-party aggregate estimation over a structured hidden database, all existing techniques rely on (an often) unrealistic assumption that the hidden database has a form-like interface (i.e., drop-down-list interface) which requires a user to enter the exact desired value for an attribute. That is, in the hidden database with drop-down-list interface, by entering a value v for a drop-down-list attribute A, a user excludes all tuples t with t½A6¼ v from the returned result.

The limitation placed by the checkbox interface prevents the traditional hidden-database aggregate-estimation techniques from being applied. Specifically, if one considers a feature (e.g., basement in [4]) as a Boolean attribute, then the checkbox interface places a limitation that only TRUE, not FALSE, can be specified for the attribute. As a result, it is impossible to apply the existing techniques which require all values of an attribute to be specifiable through the input Ib interface.

It is important to note that, in addition to the checkboxattribute-specific limitation stated above, such databases also have the same limitations as the (traditionally studied) hidden databases with drop-down-list interfaces—i.e., (1) a top-k restriction on the number of returned tuples, and (2) a limit on the number of queries one can issue (e.g., per IP address per day) through the Ib interface.

1.2 Outline of Technical Results

In this paper, I develop three main ideas for aggregate estimation over the hidden databases with checkbox interfaces:

UNBIASED-EST. I start by showing a unique challenge imposed by the hidden databases with checkbox interfaces. Note that a common theme of the existing analytic techniques for hidden Ib databases is to first build a many-to-one mapping from all tuples in the database to a set of pre-defined queries (in particular, a query tree [5]), and then draw sample tuples (from which an aggregate estimation can be made) through sampling the query set. For the hidden databases with checkbox interfaces, hoIver, it is impossible to construct such a query tree because, unlike the hidden databases with drop-down-list interfaces, it is impossible to pre-compute a set of non-overlapping queries which guarantee to return all tuples in this kind of hidden database. As a result, one has to rely on a set of overlapping queries to support aggregate estimation (e.g., through a query-sampling process)—which may lead to biased results because different tuples may be returned by different numbers of queries (and therefore retrieved with different probabilities). Our first idea is to organize these overlapping queries in a left-deep-tree data structure which imposes an order of all queries. Based on the order, I are capable of mapping each tuple in the hidden database to exactly one query in the tree, which I refer to as the designated query. By performing a drill-down based sampling process over the tree and testing whether a sample query is the designated one for its returned tuple(s), I develop an aggregate estimation algorithm that provides completely unbiased estimates for COUNT and SUM queries.

IIGHTED-EST. The error of an aggregate estimation consists of two components: bias and variance.[1] Given that our first idea guarantees zero bias, I develop our second idea of Iighted sampling to minimize variance. Specifically, I dynamically adjust the probability of sampling a query based on the query ansIrs I receive so far, in order to “align” the sampling process to both the data distribution and the aggregate to be estimated, and thereby reduce the variance of our aggregate estimations.

CRAWL. Finally, I find that certain tuples in a hidden database—specifically, lowly ranked tuples that can only be returned by queries with a large number of conjunctive predicates—can cause a significant increase in the variance of aggregation estimations. To address the problem, I develop a special-case handling procedure which crawls such tuples to significantly reduce our final estimation error.

I combine the three ideas to develop Algorithm UNBIASED-IIGHTED-CRAWL, which produces unbiased (for COUNT and SUM) aggregate estimations with small variances. Our experiments on both synthetic and realworld data sets confirm the effectiveness of UNBIASEDIIGHTED-CRAWL over various data distributions, the number of tuples and top-k restrictions.

The main contributions of this paper can be summarized as follows:

I introduce a novel problem of aggregate estimations over the hidden Ib databases with checkbox interfaces, and outline the unique challenges it presents, which prevent the traditional hidden-database-sampling techniques from being applied.

To produce unbiased aggregate estimations over the hidden databases with checkbox interfaces, I develop the data structure of left-deep-tree and define the concept of designated query to form an injective mapping from tuples to queries supported by the Ib interface.

To reduce the variance of aggregate estimations, I develop the ideas of Iighted sampling and specialtuple-crawling.

Our contributions also include a comprehensive set of experiments which demonstrate the effectiveness of our UNBIASED-IIGHTED-CRAWL algorithm on aggregate estimation over real world hidden databases with checkbox interface, as Ill as the effectiveness of each of our three ideas on improving the performance of UNBIASED-IIGHTED-CRAWL.

2 PRELIMINARIES

2.1 Model of Hidden Databases with Checkboxes

In most parts of the paper, we focus on the case where a hidden database consists solely of checkbox attributes. We shall show an easy extension of our results to the general hidden databases (i.e., with both drop-down-list attributes and checkbox attributes).

Let D be a hidden database with m checkbox attributes A_1;...;A_m and n tuples. Each checkbox attributes has two values 0 and 1,but only predicates of the form A_i ¼ 1 are allowed because of the restriction of the checkbox interface. The typical interface where users can query is by specifying values of a subset of attributes. Now suppose a user selects checkboxes A_{i1};...;A_{ij} from the interface.

TABLE 1

Running Example

tid	A	B	C
t1	0	0	1
t2	0	1	0
t3	1	0	1
t4	1	1	0
t5	1	1	1

values 0 and 1, but only predicates of the form A_i ¼ 1 are allowed because of the restriction of the checkbox interface.

The typical interface where users can query is by specifying values of a subset of attributes. Now suppose a user selects checkboxes A_{i1};...;A_{ij} from the interface. With such selections, the user constructs a query with A_{i1} ¼ 1;...; A_{ij} ¼ 1. We present the query q by the following SQL statement:

SELECT FROM D WHERE A_{i1} ¼ 1 AND AND A_{ij} ¼ 1,

which we denote ^{as} fA_{i1} & ... & A_{ij}gq or directly fA_{i1};...;A_{ij}gq in the later part of the paper for the sake of simplicity. Notation fgq represents a query with no attribute being checked.

The hidden database will search for all tuples, which we refer to as SelðqÞ, satisfying the user-specified query. There are in total jSelðqÞj tuples satisfying q, but only min fjSelðqÞj;kg tuples can be returned to the user, where k is as in the top-k restriction. We assume that these tuples are returned according to static ranking functions [6] which ensure that the order of any two returned tuples t_i and t_j won’t change by issuing different queries.

We classify queries into the following three categories, depending upon the number of tuples a query q matches and the top-k restriction:

jSelðqÞj¼ 0, this query is underflow. There is no results returned.

0 < jSelðqÞj k, this query is valid. All results are returned within top-k.

jSelðqÞj > k, this query is overflow. Only the top-k tuples can be returned together with an overflow flag.

2.2 A Running Example

We use a running example to show the previously defined model of the hidden database with checkbox interfaces. Consider a simple hidden database D with three checkbox attributes A;B;C and 5 tuples t_1;...;t_5. The hidden database is shown in Table 1, where tid is the tuple's id other than an attribute in the application level. We assume that the top-k restriction is k ¼ 2. The static ranking function is according to the subscript of tid from small to large order. Suppose a user, in this running example, selects the attribute A as his/her query. The corresponding SQL statement is,

SELECT FROM D WHERE A ¼ 1

We can see that tuples t_3;t_4;t_5 match this query in the hidden database. But with the top-k restriction, only 2 tuples, t_3 and t_4, can be returned to the user with an overflow flag.

2.3 Problem Definition

Any query of a hidden database with a checkbox interface can be represented into a SQL statement as:

SELECT * FROM D WHERE A_{i1} ¼ 1 AND AND A_{ij} ¼ 1, where D is a hidden database. A_{i1} ¼ 1;...;A_{ij} ¼ 1 indicate that the user has checked attributes A_{i1};...;A_{ij} through its checkbox interface. However, many applications may need to perform aggregate queries which are not provided by the hidden database. For example, a user may want to know the total number of cars with navigation systems, or the total prices of all cars in a car database. The formal definition of the problem is as follows.

Given a query budget G and an aggregate query Q: SELECT AGGRðÞ FROM D WHERE A_{i1} ¼ V_{i1} AND AND A_{ij} ¼ V_{ij}, where AGGRðÞ is COUNT, SUM or AVG, and V $_{i1}$;...;V_{ij} 2f0;1g are values specified for checkboxes, minimize the mean square error MSEð^{Q}^Þ¼ E½ð^{Q}^ QÞ^{2}of for estimating Q while issuing at most G queries.

One brute force solution to the problem is to compute the aggregate values over all returned tuples which are gathered by exhausting all possible checkbox queries provided by the hidden database. However, it is impossible in many situations due to the huge query cost required. In this paper, we are going to solve the problem by estimating aggregate values (COUNT, SUM) through sampling techniques.

In this paper, we use COUNT(*) as the thread to address our technical solution and the extension to other types of aggregate queries can be found in Appendix J, which can be found on the Computer Society Digital Library at http:// doi.ieeecomputersociety.org/10.1109/TKDE.2014.2365800.

2.4 Performance Measures

We consider the following two performance measures.

The accuracy of generated estimations. We use the relative error to indicate the estimation accuracy. Consider an estimator ^u used to estimate an aggregate query with real answer u, the relative error of ^u is defined as

relErrð^uÞ¼ðj^uujÞ=u: (1)

The number of queries issued through the web search interface. To measure the efficiency of the aggregate estimation, we count the total number of distinct queries issued for aggregate estimation as the query cost. The reason for using such an efficiency measure is because many real-word hidden databases may have Per-IP/user limitation such that the system may not allow one user to access the system too many times in a given period. We aim to achieve less relative error using less query cost.

2.5 Tables of Notations

The notations used in the paper are shown in Table 2.

3 ESTIMATION ALGORITHM

In this section, we develop our first idea, an unbiased COUNT estimator for the hidden databases with checkbox interfaces. We first start by bringing the idea of hidden database sampling.

TABLE 2
Table of Notations

Notation	Meaning
D	the hidden database
$A_1 A_m$	attribute set in D
$t_1 ....t_n$	tuple set in D
m	the number of attributes
n	the actual number of tuples in D
n_i	the number of tuples with designated queries at level i in the left-deep tree
n~	the estimated number of tuples in D
k	the maximum number of tuples returned in a query
V path	the path space
q	a query
jqj	the number of tuples returned in the query q
dðqÞ	tuples with designated queries being q
nðqÞ	tuples with designated queries being nodes in the subtree under q
nðA_i;A_jÞ	tuples containing both A_i and A_j
fA_i & $A_j$gq	A query, constructed by the specified attributes A_i and A_j
q^0 ¼ q & A_i	A query q^0 containing all attributes in q and an additional attribute A_i
q^0 ¼ q A_i	A query q^0 containing all attributes in q except the attribute A_i
Selt½A	the value of A_i in tuple t equals to 1
ð^i_qÞ¼ 1	tuples satisfy the query condition of q

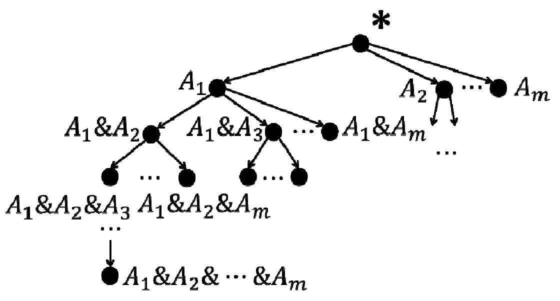

Fig. 1. A left deep tree.

problem is that a tuple may also be returned by other queries. Only taking the sampling probability of one query as the sampling probability of a returned tuple will bring bias to the estimation.

To solve this bias, it is critical to build a proper mapping between queries and tuples such that we can derive the probability for a tuple to be sampled from the probability we used to sample the query which returns the tuple. We assign a tuple to exactly one query, called designated query, which is essentially a one-to-many mapping from queries to tuples, i.e., a tuple can be designated to one and only one query, while a query may designate multiple tuples. With such a mapping, one can see that the probability for sampling a tuple can be easily derived from the sampling probability for the query that returns it—specifically, if jdðqÞj is the total number of the tuples designated by q, then the probability for sampling a tuple t returned by q is pðqÞ=jdðqÞj, where pðqÞ is the probability to sample query q. Nevertheless, in order for this idea to work, one has to address two problems: (1) how to define a rule which can assign a multiply returned tuple to only one query? and (2) how to check whether a given query is the designated one for a tuple using less or no additional query cost? Keeping this in mind, we address our solutions in the next section.

3.2 Designated Queries

To simplify our discussions, without loss of generality, we require that attributes in each query fA_{i1} & A_{i2} && A_{ih}gq be put in alphabetic order according to the index of attributes, such that $i_1 < i_2 << i_h$. Then we define the order of q following the order of subscript i:

orderðqÞ¼ alphabeticð$A_{i1}A_{i2}$ ...A_{ih}Þ; (2)

where function alphabeticðstringÞ outputs string in alphabetic order. Thus, under this definition, any query is transformed into the corresponding string of its attributes ordered alphabetically. Then, we have

Definition 1. For any two queries q_1 and q_2, q_1 q_2 if and only if orderðq_1Þ orderðq_2Þ. We call that q_1 precedes q_2 or q_2 succeeds q_1.

The above definition gives a complete order over the query set of our hidden database. For example, queries q_1 ¼fA_1gq, q_2 ¼fA_2gq, and q^1¼fA_1 & A_2gq, are ordered as q_1 q_3 q_2. With

[1].1 Hidden Database Sampling and Left-Deep Tree To estimate the size of a hidden database, one intuitive idea is to perform tuple sampling. Assume that we sample a tuple t with probability pðtÞ, we can easily estimate the size of the hidden database as n~ ¼ 1=pðtÞ. However, tuples cannot be directly sampled, because they can only be accessed through the queries provided by the

this definition, we can define a rule to solve the problem caused by multiply returned tuples in the hidden database.

^{1}.1 Hidden Database Sampling and Left-Deep Tree To estimate the size of a hidden database, one intuitive idea is to perform tuple sampling. Assume that we sample a tuple t with probability pðtÞ, we can easily estimate the size of the hidden database as n~ ¼ 1=pðtÞ. However, tuples cannot be directly sampled, because they can only be accessed through the queries provided by the hidden database. Therefore, we transform the tuple sampling to query sampling.

Definition 2 [Designated query]. Suppose a tuple t can be returned by queries q_1;q_2;...;q_k, which are in the order q_1 q_2 q_k, then we define q_1 as the designated query of t.

With this definition we uniquely assign a tuple which can be returned in multiple queries into one and only one query among those potential queries. Thus, we have solved the first problem caused by the resulting overlap of different queries. Nonetheless, the other problem remains—i.e., when getting to a query in the sampling procedure, how can we determine whether the query that returns a tuple is indeed its designated query.

A baseline solution is to check all queries which precede the current query to see if some of them in the most prior position also return this tuple. That will make the sampling extremely inefficient. Fortunately, with the assumption that tuples in D are returned with a static ranking function (which is mentioned in Section 2), we do not need to actually perform such heavy testing. Rather, only one additional query testing is necessary.

---

hidden database. Therefore, we transform the tuple sampling to query sampling.

Recall that D has m checkbox attributes as its query interface, one can enumerate that there are in total 2^m possible queries, from fg$_q$ to fA$_1$ && A$_m$g$_q$ which are all possible combinations of the m attributes. All of these 2^m queries are in the query space. Because if we discard any query from them, we may not be able to access to some tuples which are only returned by the discarding queries. We organize all these queries in the query space with a left-deep tree structure as shown in Fig. 1, where every node is corresponding to a query and a directed edge from a node to a child node indicates that the query corresponding to this child node includes all attributes in the parent query and one additional attribute. The root node represents query fg$_q$, while the bottom leaf A $_1$ && A$_m$ represents a query with all attributes being checked.

In the later part, we will introduce our query sampling algorithm which will be performed on this left-deep tree. Before doing so, we need to consider how we transform the probability of a query to the probability of a tuple. A straightforward way is to assign the probability of a query to the tuples which are returned by this query.

Theorem 3.1. Given a tuple t and a query q which can return the tuple t, it only takes one query to test whether q is the designated query for t.

Proof. Here we give the main ideas of the proof. If q is the designated query for t, then both of the following conditions should be satisfied. 1). For any attribute A_i, t½A_i¼ 1 and A_i 2= q, A_i cannot precede any attribute of q, otherwise q^0 ¼ q & A_i (q^0 q) returns t; 2). Queries, whose attribute sets are truncated from attribute set of q in terms of alphabetic order, should not return t. The first condition can be easily checked from t's value (without issuing queries), while the second condition only requires to check if q fA_{ih}g (here A_{ih} is the last attribute of q) does not return t. Details can be found in Appendix A, available in the online supplemental material. tu

So, for each query, we need one additional query for designated query testing of returned tuples.

We can further save this one additional query cost for designated test, if we perform a drill-down sampling on the left-deep tree.

4 VARIANCE REDUCTION

In this section, we analyze how weight allocation affects the estimation variance. We first motivate the effectiveness of variance reduction by describing an ideal weight assignment algorithm that achieves zero variance (and therefore zero estimation error) for COUNT(*) query but is impossible to apply in practice. Then, we briefly review the struc-
ture-based weight allocation method used in the UNBIASED algorithm, and discuss how to further reduce estimation variance. Specifically, we shall introduce two novel techniques, weight adjustment and low probability crawl technique, which significantly reduce the estimation variance and thereby the final estimation error for aggregate queries.

First, to ensure an unbiased estimation, a fundamental requirement is for the weight of each edge to obey the following rules:

The sum of weights of edges under one node should equal to 1;

Every edge has a non-zero weight whenever there exists a tuple with designated query being a node in the subtree under this edge.

The ideal weight allocation algorithm requires that the weights p $_1$;p_2;...;p_k of edges, from q to its children nodes q $_1$;q_2;...;q_k, are exactly proportional to the number of tuples with designated queries being nodes in the corresponding subtrees under q $_1$;q_2;...;q_k. Theorem 4.1 shows that the ideal weight allocation scheme leads to zero estimation variance (and error).

Theorem 4.1. The ideal weight allocation leads to zero estimation variance (and error).

Proof. The proof includes two steps. First, with the ideal weight allocation, we calculate the estimated value of the number of tuples for an arbitrary random drill-down path. Then, we derive that the estimated value just equals to the true value. The complete proof is shown in Appendix B, available in the online supplemental material. tu

The ideal weight allocation algorithm is impractical, because the underlying data distribution used for computing the probabilities cannot be known beforehand in the hidden database.

4.1 Structure-Based Weight Allocation

Since one does not have knowledge of the underlying data distribution in practice, the UNBIASED algorithm uses a (over-)simplified assumption that all attributes are mutually independent, and having uniform distribution (over {0, 1}). Then the number of tuples that have been designated by nodes in a subtree is proportional to the number of nodes in this subtree. With the above assumption and Theorem 4.1 , we assign the weights of edges corresponding to the number of nodes in their pointed subtrees. With the left-deep tree structure, suppose a node q has j children q_1;q_2;...;q_j from left to right. Then the proportion of edge weights under q from left to right should be pðq1jqÞ : pðq2jqÞ : : pðqjjqÞ¼ 1=2 : 1=4 : : 1=2j : (6)

After normalization, we can determine the probability of each edge of the left-deep tree. This weight allocation is used in the UNBIASED algorithm.

Unfortunately, the independence-and-uniform assumption rarely fits in practice. As a result, UNBIASED estimation algorithm often leads to an extremely large estimation variance (and therefor, estimation error). Recall that in Example 1, the variance is mainly caused by the difference between the fixed probability allocation and the exact probability distribution. We shall propose an automatic weight adjustment algorithm to significantly reduce the estimation variance in next section.

4.2 Automatic Weight Adjustment

From Theorem 4.1, we know the variance of the estimator can be significantly improved if weight allocation of edges in the left-deep tree is proportional to the data distribution of subtrees linked by those edges. When conducting drill-down sampling, an increasing number of tuples are gathered. Therefore, it is intuitive to use those valuable tuples to learn the data distribution in the hidden database. More specifically, using q as a query, we want to make an estimation of the total number of tuples, denoted as nðqÞ, having nodes in the subtree under node q as theirs designated queries.

For a query q, let nðqÞ and dðqÞ be the set of tuples with designated queries being nodes of the whole subtree under q (including q itself) and the set of tuples with designated queries being the only node q respectively, then, jnðqÞjjdðqÞj is the number of tuples with designated queries being all node in the subtrees under the children nodes of q. Now, let q ¼fA$_{i1}$ && A$_{ik}$gq (attributes in alphabetic order), according to the definition of the leftdeep tree, q should have l ¼ m k children as q$_1$ ¼ q & A$_{j1}$;q$_2$ ¼ q & A$_{j2}$;...;q$_l$ ¼ q & A$_{jl}$ where A$_{j1}$ð¼ A$_{ik}$þ1Þ;
A$_{j2}$;...;A$_{jl}$ð¼ A$_m$Þ are all those attributes succeeding A$_{ik}$
(the last attribute of q). Then, we have:

1
XjnðqiÞj¼jnðqÞjjdðqÞj; (7)
i¼1

where q$_i$, i ¼ 1;2;...l, are children of q, jnðq$_i$Þj is the number of tuples having nodes in the subtree under q$_i$ as theirs designated queries.

For the ith branch q$_i$ under node q, q$_i$ ¼ q & A$_{ji}$ ¼ fA$_{i1}$ && A i$_k$ & A$_{ji}$gq. One can see that any tuple t 2 nðq$_i$Þ has the following properties according to the definition of designated query.

The values of those attributes which appear in query q$_i$ should be equal to 1, otherwise it cannot be returned in q$_i$. So we have t½A$_{i1}$¼ 1;...;t½A$_{ik}$¼ 1, and t½A$_{ji}$¼ 1.

For any attribute A A$_{ji}$ and A 2= q, t½A¼ 0 (or else t should not be designated by nðq$_i$Þ). Let fA$_{s1}$;
A$_{s2}$;...;A$_{sv}$g be all such attributes.

Based on the above properties, for any tuple t 2 ðnðqÞndðqÞÞ, the probability that t 2 nðq$_i$Þ satisfies:

pji / pðAs1 ...AsvAi1 ...AikAjiÞ; (8)

where p$_{ji}$ is the weight for the edge from q to q$_i$, $^A{}_k$ is for A$_k$ ¼ 0, and A$_k$ is for A$_k$ ¼ 1. To save our notations, we use
A$_k$ to represent either A$_k$ or $^A{}_k$, then we have
p$_{ji}$ / pA$_1$ ...A$_j$i: (9)

For each branch q $_i$ ¼ q & A$_{ji}$, we have jnðq$_i$Þj¼ p$_{ji}$ ðjnðqÞjjdðqÞjÞ, where Pli¼1 pji ¼ 1.

After normalizing p$_{j1}$;...;p$_{jl}$, we get the weights (probabilities) of all edges under node q. This allocation is derived from gathered tuples along our sampling, which can well approximate the real data distribution of the hidden database.

Given the scheme in Equation (9), one still has to estimate the joint distribution of pðA$_1$ ...A$_{ji}$Þ. In this paper, we consider two estimation methods, attribute independent model and attribute dependent model, respectively.

4.2.1 Attribute Independent Model

We start with a (somewhat cruel) approximation of the joint distribution by the simple multiplication of the marginal probability of each attribute (i.e., following the attributeindependent assumption). Then, Equation (9) can be decomposed as

ji
pA_1 ...A_{ji}¼YpA_k: (10)
k¼1

For those A_k ¼ $^A{}_k$, pð$^A{}_k$Þ¼ 1 pðA_kÞ.

Recall the working principles of UNBIASED algorithm for the count estimation of D. The number of tuples for A_k ¼ 1 can also be estimated in the similar way at the same

time.
n^eðAof$_k$Þ¼designatedP$^h{}_{i¼}$0ðjdðq$_{i;j}$tuplesÞjA$_k$=pcontainingðq$_{i;j}$
ÞÞ, whereAj$_k$dðinq$_{i;j}$ÞjnodeA$_k$ is qthe $_{i;j}$.
number
Now we denote the currently estimated number of tuples in D and the estimated number of tuples with A_k ¼ 1 as n and nðA_kÞ respectively. Then, we can approximate the probabile itye pðA_iÞ as, e

pðA_kÞ¼ nðA_kÞ=n: (11) e e e

Our algorithm UNBIASED-INDEPENDENT conducts the aggregate estimation using two phases. In the first phase, UNBIASED algorithm is executed to perform drilldown sampling with structure-based weight allocation scheme on the left-deep tree. At the same time, visited tuples are gathered into a set T. In the second phase, we use T to compute pðA $_i$Þ, for i ¼ 1 to m, and call INDEPENDENT weight allocatione to adjust weights of edges. Then, drill-down sampling algorithm is performed with the updated weight allocation of edges, and T is also updated with newly gathered tuples.

Actually, the second phase will be recursively executed until the query cost exceeds the query budget. There is a pre-determined pilot budget w to separate the above two phases, where w is the number of queries. When the number of queries exceeds w, the algorithm estimates pðA_iÞ and adjusts weight allocation accordingly, then performse new drill-down sampling.

In Example 1, we add weight adjustment to the estimation algorithm. When weight adjustment begins, suppose we estimate p~ðA_iÞ as 0.01. If a drill-down path happens to go through the path from the root node to node A_1, then the estimated number of tuples by this path is 210 (i.e., 10+2/ 0.01) which is much closer to the exact number of tuples (=200).

4.2.2 Attribute Dependent Model

UNBIASED-INDEPENDENT algorithm is based on the assumption that attributes are mutually independent. In the real world, attributes of a hidden database are often correlated with each other. Take the hidden database Car Finder as an example, if a car contains leather seats, it usually contains A/C. One can leverage such correlation to improve the performance of drill-down sampling algorithm.

In this section, we study a more general case where correlations among attributes may exist. Therefore, we cannot simply decompose Equation (9) into individual attribute distributions. Rather, it should be computed with consideration of the correlations between attributes. To compute the joint probability of attributes pðA_{i1} ...A_{ij}Þ, a simple method is pðA_{i1} ...A_{ij}Þ¼ nðA_{i1} ...A_{ij}Þ=n, where nðA_{i1} ...A_{ij}Þ is the $^{e\ e\ e\ e}$
estimated number of tuples satisfying t½A_{i1}¼ A_{i1};...; t½A_{ij}¼ A_{ij}.

With m attributes, there are as many as 2^m joint probabilities which need to be estimated. It is extremely hard, if not impossible, to estimate all those joint probabilities along our sampling estimations. In fact, this problem has been well studied in the past work and some general solutions can be found in reference [7]. In practice, with our close study and preliminary experiment, considering of correlations among multiple (more than two) attributes may not have significant improvement of the estimation performance in most of real-world applications. To save the cost and simplify the computation, in this paper we only check and make use of correlations between two attributes in the joint probability estimations.

To see which two attributes may have potential correlations, we apply x^2-test [7] on tuples gathered previously. The value x^2 of the test statistic is thus calculated. The bigger the value of x^2 is, the stronger evidence is against the independence hypothesis between the two attributes. A p-value is used for measuring how much we have against the independence hypothesis. The smaller the p-value is, the more evidence we have against independence hypothesis. A pair of attributes are correlated if p-value is under a threshold which we use 0.01 in our experiment.

To compute p_{ij} in Equation (9), we use the correlations between two attributes as below:

pA_1 ...A_{ji}¼Yp$A_u$$A_vYpA_t$; (12)

where A_u and A_v are a correlated pair of attributes, and A_t represents independent attributes. In our algorithm, pðA_t Þ is computed the same as in Equation (11), and pð$A_u$$A_v$Þ can be estimated as pð$A_u$$A_v$Þ¼ nð$A_u$$A_v$Þ=n.

To find whiche pairwise ecombinationse are correlated, we perform the x^2-test on all possible pairwise combinations of attributes once at the time weight adjustment begins. We also can perform it after each drill-down sampling when the number of gathered tuples increases, and nðA$_u$A$_v$Þ is estimated together with n at the same time. Wee only consider pairwise correlations ein the computation of probability for multi-attribute combinations. Thus if one attribute (e.g., A$_i$) is correlated to many other attributes (e.g., A$_{j1}$;A$_{j2}$;...;A$_{jk}$) , then we only take one pair of them which has the smallest p-value into account. An example is considering the following correlated pairs of attributes: (A$_1$;A$_2$), (A$_1$;A$_3$), (A$_2$;A$_3$) , (A4;A5), then pðA1A2A3A4A5Þ¼ pðA1A2ÞpðA3ÞpðA4A5Þ.

We are now ready to combine the weight assignment algorithm with our drill-down method to produce UNBIASED-WEIGHTED. One can see the difference between UNBIASED-WEIGHTED and UNBIASED-INDEPENDENT lies in the second phase (i.e., probability computation). Specifically, at the beginning of the second phase, the x^2-test is used to find the pairs A$_u$ and A$_v$ which are correlated. Besides estimating pðA$_i$Þ, we also need to estimate pðA$_u$A$_v$Þ using the gathered tuplee set T. This estimation is alsoe recursively conducted.

Recall that in Example 1, there is an assumption that all attributes are independent. Now suppose attributes A$_2$ and A$_3$ are correlated. The relationship between them is that for each tuple, the value of A$_3$ equals to the value of A$_2$. With the attribute independent model, the expected value for pðA2A3Þ is E½pðA2A3Þ¼ E½ð1 pðA2ÞÞ pðA3Þ¼ð1 1=2Þ 1e=2 ¼ 1=4. Whilee with attributee dependente model, the expected value for pð$^A{}_2$A$_3$Þ is 0 which equals to the exact probability. The moree accurate of the estimated probability, the much closer of the weight allocation to the ideal weight allocation, which leads to less estimation variance.

4.3 Low Probability Crawl

From the above discussion, our weight adjustment algorithms can effectively reduce the variance in many cases such as in Example 1, but there are still cases where the drill-down sampling may produce high estimation variance, as illustrated by the following example.

Example 2. Consider the hidden database D with 10 attributes, and 11 tuples. The top-k restriction is k ¼ 10. These tuples are returned with the order t$_1$;...;t$_{11}$. Tuple t$_{11}$ is ð$^A{}_1$;...;$^A{}_9$;A$_{10}$Þ¼ð0;...;0;1Þ, which only contains the last attribute A$_{10}$. For tuples t$_1$ ...t$_{10}$, all attributes are independent and have uniform distributions (over {0, 1}).

In this hidden database, tuples t $_1$;...;t$_{10}$ are returned and designated by the root node. Tuple t $_{11}$ is returned and designated by the right-most node fA $_{10}$gq. The probabilities of edges under the root node are 1=2, 1=2^2, 1=2^3, ..., 1=2^{10}. For the right-most drill-down path, the estimated number of tuples is 1,034 (i.e., 10 þ 2^{10}). For other drill-down paths, each estimated number of tuples is 10. Then the estimation variance is Oð2^{10}Þ which is mainly caused by the right-most drill-down path.

In Example 2, if we crawl the whole tree which means we issue all nodes that may designate a tuple, then we can get the exact number of tuples with only nine more queries comparing to a random drill down.

In a general case, we crawl the whole subtree under node q (e.g., the root node in Example 2) and assign all tuples with designated queries being nodes in the subtree of q to node q. This method is called the Low Probability Crawl. It not only avoids estimation variance caused by a tuple designated by a deep node, but also obtain the exact number of tuples with designated queries being nodes in the subtree of q with a few queries. A crawling threshold c is used to trigger this processing.

The detail procedure of the Low Probability Crawl is as follows. For each drill-down path, if the probability of an overflow node q is less than the crawling threshold c, then we issue all queries under this node. There are in total jnðqÞj tuples being designated to node q. Then for the drill-down path with low probability crawl, Equation (3) is changed to

¼ h ¼ h1 jdððqiÞÞjþjnððqhÞÞj ——— ———
n~ Xi¼0 n~i Xi¼0 p qi p qh : (13)

Theorem 4.2. Equation (13) is an unbiased estimator for the number of tuples in D.

Proof. The proof is similar to the proof of Theorem 3.2. See Proof of Theorem 4.2 in Appendix I, available in the online supplemental material. tu

We embed Low Probability Crawl to every drill-down path in the UNBIASED-WEIGHTED algorithm to get a new algorithm called UNBIASED-WEIGHTED-CRAWL. It not only keeps the unbiasedness of the algorithm, but also reduces the risk of reaching a low level node with extremely small probability for the estimation, thereby significantly reduces the variance and final estimation error.

5 EXPERIMENT

In this section, we describe our experimental setup and conduct evaluations of our proposed algorithms UNBIASED, UNBIASED-INDEPENDENT, UNBIASED-WEIGHTED, and UNBIASED-WEIGHTED-CRAWL. We also compare the performance of the algorithms with different parameter settings.

5.1 Experiment Setup

1) Hardware and platform. All experiments were conducted on an Intel Xeon E5620 2.40 GHz CPU machine

with 18 GB memory. All algorithms were implemented in Java.

2) Data sets. To evaluate the performances of our algorithms on different data distributions, we generated three kinds of synthetic data sets, each of which was with 20 attributes and contained in total 10,000 tuples as the default count, but with different attribute distributions. Further, we also performed our algorithms on a real data set which was crawled from a publicly available commercial hidden database.

i.i.d synthetic data set. This data set was generated as independent and identical distribution of attributes. Let A_i, i ¼ 1...m are attributes of the data set, pðA_iÞ is the probability for A_i ¼ 1. Here we set pðA_iÞ¼ 0:1 for i ¼ 1 to m in our experiments.

Skew-independent synthetic data set. The second data set was generated as skewed, but still independent. In other words, for different A_i, pðA_iÞ had much different values. But they were still generated independently. In this paper, for attributes A_1 and A_2, we set pðA_1Þ¼ pðA_2Þ¼ 0:1. For A_3 to A_m, we set pðA_3Þ¼ 1=90;pðA_4Þ¼ 2=90;...;pðA_mÞ¼ m=90 with a step of 1=90, where m is the number of attributes.

Skew-dependent synthetic data set. The third data set was generated as skewed and dependent. Some of the attributes had correlations. That is, in our experiments, in order to make A_1 and A_2 have correlations, we enforced pðA_1 ¼ A_2Þ¼ 80% in generating the data set. Therefore the only difference between skew-dependent and skew-independent data set is that A_2 has 80 percent probability to have the same value as A_1 in skew-dependent.

Real data set. The real data set was called job-search, which was crawled from the website [2]. It took more than one week to crawl the data from 28th March to 6th April in 2011. There are 95 attributes and in total 109,487 tuples in this dataset. Attributes are various kinds of industries (e.g., ”Banking”), job type (e,g,. “Full Time”), education levels (e.g., from ”Student (High School)” to ”Senior Executive (President, CFO, etc)”), categories (e.g., “Creative/Design”) etc. The most frequent attribute is “Full Time” which is contained in 81,766 tuples. While the least frequent attribute is “Performing and Fine Arts” which is contained in 95 tuples. We should notice that, for some hidden databases including website [2], if no or a very few tuples satisfying the query condition, instead of returning empty or a very few answers, they relaxes the search condition to return some tuples which satisfy some of the attributes. But with a local filter, we can still extract the tuples exactly satisfying the query condition.

3) Aggregate estimation algorithms. We tested the four proposed algorithms:

UNBIASED. This algorithm is the baseline estimation method under the assumption that designated tuples are well distributed among all query nodes.

UNBIASED-INDEPENDENT. This algorithm incrementally adjusts the drill-down weights after w queries. The weight assignment follows an assumption that the attributes are independent.

UNBIASED-WEIGHTED. This algorithm assumes attributes may be dependent, and perform weight adjustment in consideration of correlations.

UNBIASED-WEIGHTED-CRAWL. This algorithm is the same with UNBIASED-WEIGHTED except crawling the whole subtree of a node if the probability of the node is lower than a given threshold.

We evaluated the scalability of the algorithms and the impacts of the parameters with various parameter settings.

4) Performance measures. We measure the performance of our algorithm in terms of their query costs and estimation accuracies. For the query cost, we counted the number of queries issued to the hidden database. The relative error was used to indicate the accuracy of the estimation.

5.2 Experiment Results

Skew-dependent was the default synthetic data set used for evaluating the main algorithm UNBIASED-WEIGHTEDCRAWL, with k ¼ 50 for the top-k restriction. For the real data set, we set k ¼ 100. For all algorithms, the default pilot budget for weight adjustment was w ¼ 100. The default crawling threshold was c ¼ 10^6 unless otherwise specified.

1) Performance of UNBIASED-WEIGHTED-CRAWL. UNBIASED-WEIGHTED-CRAWL is the most advanced algorithm proposed in this paper. Specifically, we evaluated

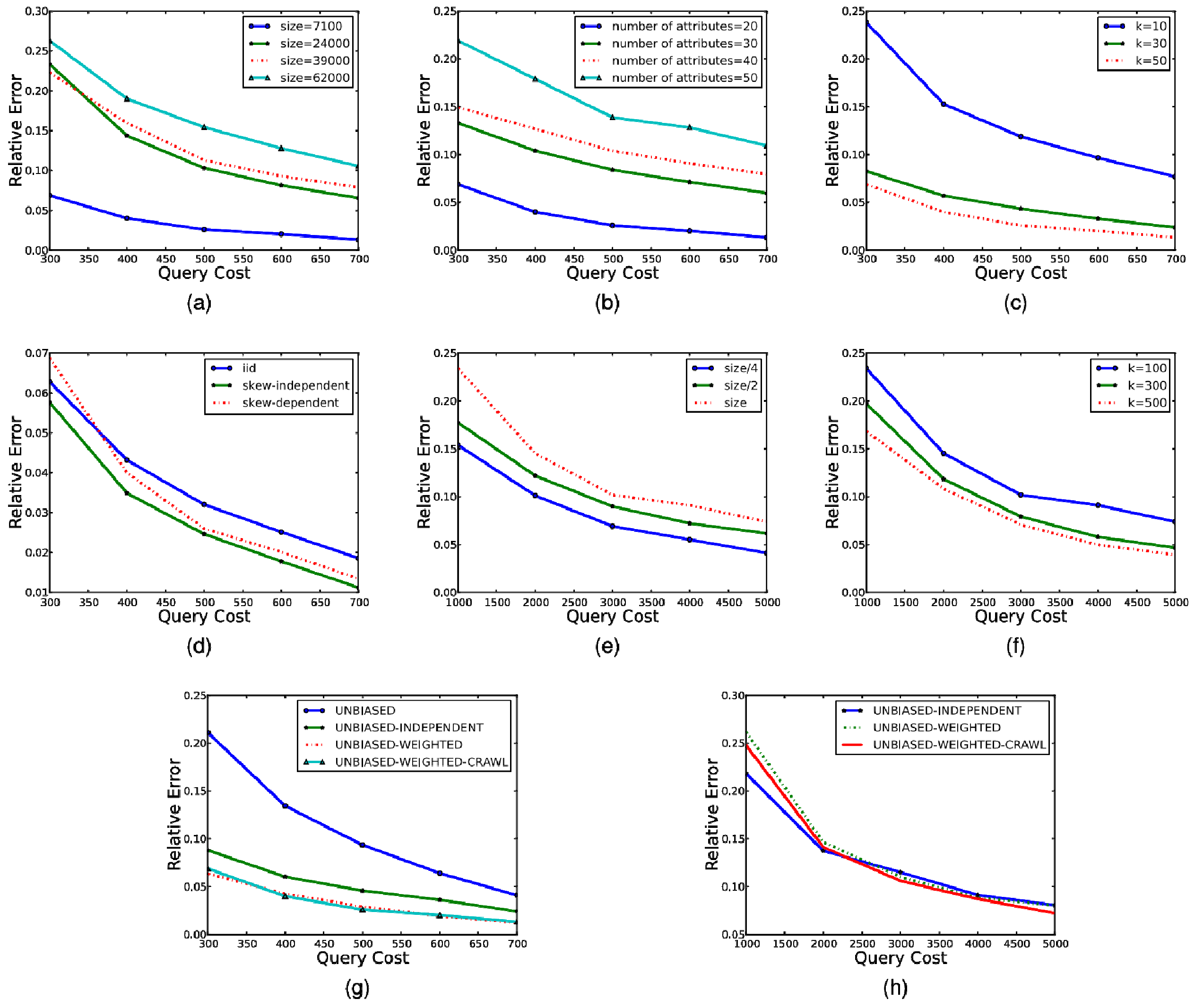

Fig. 3. Experiment results: (a) Impact of size on synthetic data set; (b) Impact of the number of attributes on synthetic data set; (c) Impact of k on synthetic data set; (d) Impact of data distribution on synthetic data set; (e) Impact of size on real data set; (f) Impact of k on real data set; (g) Impact of algorithms on synthetic data set, and (h) Impact of algorithms on real data set.

its scalability over different data distributions and parameter settings.

Different Counts of D. We first tested the algorithm UNBIASED-WEIGHTED-CRAWL with different number of tuples n. For this purpose, we generated four skew-dependent data sets with 10,000, 20 000, 50 000, and 100 000 tuples respectively. Because of the query capability and the top-k restrictions, the database cannot return all generated tuples. We only cared about the number of returned tuples, and the number of returned tuples from the above four data sets are 7;100, 24;000, 39;000, and 62;000 respectively. Fig. 3a shows the tradeoff between relative error and query cost when n varies from 7;100 to 62;000. With fixed cost, the larger the data sets, the more the relative error. The reason is easy to understand that more tuples may lead to a high possibility of a tuple having a deep node as its designated query. If the end node is deep in the drill-down path, which means there are more nodes used in a single drill-down path, then the total number of drill-down paths will be reduced since the fixed cost. Each drill-down path can be taken as a sample in computing the final results. The smaller of the sample count, the bigger of the variance. Another possible reason is that variance of each node in the drill-down path will be accumulated, the probability of a deep node is likely far away from the real probability.

Different number of attributes. To know the impact of the number of attributes on the UNBIASED-WEIGHTEDCRAWL, we generated four skew-dependent synthetic data sets with 20;30;40 and 50 attributes respectively. Fig. 3 b illustrates the relationship between relative error and query cost for different number of attributes. One can see that the relative error decreases dramatically when the number of attributes decreases. It is reasonable because more attributes lead to more complicated correlations between them. In our algorithm, the probabilities of edges are simply reduced to only consider correlations between two attributes, while the real correlations may be much more complicated. The limitation of our algorithm in finding the complicated correlations leads to more variance.

Different Top-k. Another parameter is the value of k in the top-k restriction. We tested UNBIASED-WEIGHTEDCRAWL with k ¼ 10;30;50, respectively. The relative error plotting query cost is shown in Fig. 3c. Larger k can lead to a better estimation, because a drill-down path can stop earlier, thus saves the cost. Given the fact that the cost is fixed, the number of queries used in one drill-down path become less. Therefore, it increases the total number of drill-down path with fixed cost, so the variance can be reduced accordingly.

Various data distributions. To know the performance of the algorithm on different data distributions, we performed UNBIASED-WEIGHTED-CRAWL on i.i.d, skew-independent, and skew-dependent respectively. Fig. 3d shows the tradeoff between relative error and query cost. For i.i.d data set, there is no correlation between attributes. In such a situation, UNBIASED-WEIGHTED-CRAWL is shrunken into UNBIASED-INDEPENDENT-CRAWL. For skew-dependent data set, there is a correlation between attribute A_1 and A_2. From the result we know that the performance of the algorithm on different data distributions is much similar, which means our proposed algorithm can work adaptively on different hidden databases.

Count sensitiveness on real data set. After testing on synthetic data sets, we tested our proposed algorithm UNBIASED-WEIGHTED-CRAWL on the real data set with different number of tuples. In order to obtain a group of real data sets with different number of tuples, we performed uniform sampling on the real data set. Besides the original real data set, we obtained two additional data sets from the original one such that one of the two was the half tuples of the original set, and another one was just a quarter of the original one. Fig. 3e shows the performance difference of the algorithm over the three real data sets. It also shows that small databases need less cost for estimation.

Top-k sensitiveness on real data set. We also tested UNBIASED-WEIGHTED-CRAWL on the real data set with different k, k ¼ 100, 300 and 500, for top-k restriction. Fig. 3f gives the relationship between the relative error and the query cost with different k. Once again, it reveals the fact that larger k can allow less cost for the estimation.

2) Performance of different algorithms. Besides evaluating the scalability of UNBIASED-WEIGHTED-CRAWL on different parameters, we also compared it with other algorithms UNBIASED, UNBIASED-INDEPENDENT, UNBIASED-WEIGHTED, which were proposed in this paper.

On synthetic data sets. We compared the above algorithms on the default synthetic data set. Fig. 3g depicts the tradeoff between relative error and query cost using different estimation algorithms. The UNBIASED algorithm has the worst performance among all. With weight adjustment, the estimation accuracy can be improved dramatically. With taking into consideration of correlations in the data set, UNBIASED-WEIGHTED shows better performance over UNBIASED-INDEPENDENT which simply assumes that attributes were independent. For the algorithm UNBIASED-WEIGHTED-CRAWL, at the very beginning of the sampling process, the drill-down algorithm is unlikely to encounter many rarely-hit tuples. As a result, we are unlikely to observe any substantial benefit of CRAWL (especially given that it also costs more queries). Nonetheless, with more queries being issued, it becomes more and more likely for UNBIASED-WEIGHTED to encounter some rarely-hit queries. As a result, the effect of CRAWL becomes more evident. Finally, when the query cost (and the number of samples obtained) becomes extremely large, both CRAWL and the baseline algorithms converge to the ground truth, leading to the non-distinguishability at the end of the curve.

On real data sets. The above algorithms had also been performed on the real dataset. Although correlations in the real dataset are not as significant as we did on the synthetic dataset, we can see that the relative error decreases when correlations are taken onto account. Fig. 3h shows that, in the very beginning, UNBIASED-INDEPENDENT performs better than UNBIASED-WEIGHTED. This is due to the inaccurate estimation of joint probabilities of attribute pairs when there are only a few sample tuples. With more and more tuples being collected by drill-down sampling, the joint-probability estimation becomes more accurate, leading to better performance of UNBIASED-WEIGHTED than UNBIASED-INDEPENDENT. For example, when the query cost exceeds 5,000, UNBIASED-WEIGHTED performs better than UNBIASED-INDEPENDENT. If we added low probability crawl to UNBIASED-WEIGHTED, then the estimation accuracy will be improved further. For example, when the query cost is 5,000, the relative error of UNBIASED-WEIGHTED-CRAWL is 0.0721 which is much smaller than the relative error 0.0804 of UNBIASEDWEIGHTED. Because in this real dataset, there exists one or two tuples designated by queries with extremely low sampling probabilities. Therefore, with Low Probability Crawl, we can decrease the estimation error.

6 RELATED WORK

Hidden databases have drawn much attention in [8], [9] , [10] recently. We now compare our work with the existing works related to querying under access limitations, cardinality estimation in database query optimization, and probing the hidden database.

Querying under access limitations.In the area related to querying under access limitations, there are a lot of works [11], [12], [13], [14] that focus on how to answer queries with binding values on some input attributes. For example, [11], [12], [13] deal with recursive query plans. Benedikt et al. [14] designs formal languages for describing the access paths that are allowed by the querying schema. All of them focus on retrieving all (or an arbitrary subset of) tuples satisfying a query under some general access restrictions. When applying these methods to answer aggregate queries in our problem, we have to retrieve all matching tuples inside the database. But it is infeasible to efficiently retrieve all tuples matching the query through a top-k interface. We recognize that the existing techniques on overcoming access limitations might be combined with the prior work on aggregate estimations over form-like interfaces (e.g., [5]) to address the problem of aggregate estimations over a checkbox interface. Specifically, since the major distinction of a checkbox interface is its disallowance of certain attribute values in query predicates (e.g., while A_1 ¼ 1 can be specified, A_1 ¼ 0 cannot), one might use the techniques for overcoming access limitations to translate a disallowed query (e.g., SELECT FROM D WHERE A_1 ¼ 0) to queries supported by the checkbox interface. With such a translation service as a middleware, the aggregate estimation techniques designed for form-like interfaces can be then used over checkbox interfaces as well. Nonetheless, while such a translation is theoretically feasible, it is often unrealistic for practical purposes, because of the large number of queries one has to translate one disallowed query into. For example, to translate the above mentioned disallowed query (A1 ¼ 0), one has to crawl about half of all tuples if data distribution is i.i.d with uniform distribution. Thus, instead of using the translation service, we studied a novel technique in this paper to address the aggregate estimation problem over checkbox interfaces directly.

Cardinality estimations in database query optimization. In the subarea of cardinality estimation for query optimization, there are two common methods. One is to maintain certain statistics of the underlying database (e.g., [15]) for the purpose of cardinality estimation, while the other is to enable cardinality estimation through sampling (e.g, [16], [17]).

7 CONCLUSIONS

Enabling analytics on hidden web database is a problem that has drawn much attention in recent years. In this paper, we address a novel problem where checkboxes exist in the web interface of a hidden database. To enable the approximation processing of aggregate queries, we develop algorithm UNBIASED-WEIGHTED-CRAWL which performs random drill-downs on a novel structure of queries which we refer to as a left-deep tree. We also propose weight adjustment and low probability crawl to improve estimation accuracy. We performed a comprehensive set of experiments on synthetic and real-world datasets with varying database sizes (from 5;000 to 100;000), number of attributes (from 20 to 50) and top-k restriction (from k = 10 to 30). We found that, as predicted by the theoretical analysis, the relative error decreases when the number of queries issued increases. In addition, for the same query budget, the relative error is lower with a smaller number of attributes and/or a large k. In the worst-case scenario, we achieve around 15 percent relative error with 500 queries issued for the synthetic dataset, and less than 10 percent relative error with about 3,500 queries issued for the real-world dataset. The experimental results demonstrate the effectiveness of our proposed algorithms.

REFERENCES

[1] C. Sheng, N. Zhang, Y. Tao, and X. Jin, “Optimal algorithms for crawling a hidden database in the web,” Proc. VLDB Endowment, vol. 5, no. 11, pp. 1112–1123, 2012.

[2] Monster, Job search page [Online]. Available: http://jobsearch. monster.com/ AdvancedSearch.aspx, 2011.

[3] Epicurious, Food search page [Online]. Available: http://www. epicurious.com/ recipesmenus/advancedsearch, 2013.

[4] Homefinder, Home finder page [Online]. Available: http://www. homefinder.com/search, 2013.

[5] A. Dasgupta, X. Jin, B. Jewell, N. Zhang, and G. Das, “Unbiased estimation of size and other aggregates over hidden web databases,” in Proc. Int. Conf. Manage. Data, 2010, pp. 855–866.

[6] A. Dasgupta, N. Zhang, and G. Das, “Turbo-charging hidden database samplers with overflowing queries and skew reduction,” in Proc. 13th Int. Conf. Extending Database Technol., 2010, pp. 51–62.

[7] A. Agresti, Categorical Data Analysis, vol. 359. Hoboken, NJ, USA: Wiley, 2002.

[8] S. Raghavan and H. Garcia-Molina, “Crawling the hidden web,” in Proc. 27th Int. Conf. Very Large Data Bases, 2001, pp. 129–138.

[9] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang, “Accessing the deep web,” Commun. ACM, vol. 50, no. 5, pp. 94–101, 2007.

[10] J. Madhavan, L. Afanasiev, L. Antova, and A. Halevy, “Harnessing the deep web: Present and future,” CoRR, vol. abs/ 0909.1785, 2009.

[11] D. Florescu, A. Levy, I. Manolescu, and D. Suciu, “Query optimization in the presence of limited access patterns,” ACM SIGMOD Rec., vol. 28, no. 2, pp. 311–322, 1999.

[12] A. Calı and D. Martinenghi, “Querying data under access limitations,” in Proc. IEEE 24th Int. Conf. Data Eng., 2008, pp. 50–59.

[13] M. Benedikt, G. Gottlob, and P. Senellart, “Determining relevance of accesses at runtime,” in Proc. 30th ACM SIGMOD-SIGACTSIGART Symp. Principles Database Syst., 2011, pp. 211–222.

[14] M. Benedikt, P. Bourhis, and C. Ley, “Querying schemas with access restrictions,” Proc. VLDB Endowment, vol. 5, no. 7 , pp. 634–645, 2012.

[15] Y. Ioannidis, “The history of histograms (abridged),” in Proc. 29 th Int. Conf. Very large data bases-Volume 29, 2003, pp. 19–30.

[16] R. J. Lipton and J. F. Naughton, “Query size estimation by adaptive sampling,” in Proc. 9th ACM SIGACT-SIGMOD-SIGART
Symp. Principles Database Syst., 1990, pp. 40–46.

[17] R. J. Lipton, J. F. Naughton, and D. A. Schneider, “Practical selectivity estimation through adaptive sampling,” Sigmod Rec., vol. 19 , pp. 1–11, 1990.