ISSN:2348-2079

Volume-5 Issue-2

International Journal of Intellectual Advancements and Research in Engineering Computations

Computer hardware purchase And sales

[1] N.Zahira Jahan [2] R.Rangeswaran
1Assistant Professor Department of MCA
2PG ScholarNandha Engineering College, Erode.
E-mail id: kingsrocky007@gmail.com

Abstract*—A large number of web data repositories are hidden behind restrictive web interfaces, making it an important challenge to enable data analytics over these hidden web databases. Most existing techniques assume a form-like web interface which consists solely of categorical attributes (or numeric ones that can be discretized). Nonetheless, many real-world web 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 website like Yahoo! Autos. We find that, for the purpose of data analytics, such checkbox-represented attributes differ fundamentally from the categorical/numerical ones that were traditionally studied. In this paper, we 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.

Index Terms—Hidden databases, checkbox, aggregate estimation, weight adjustment

1 INTRODUCTION

Hidden databases are data repositories ”hidden behind”—i.e., only accessible through—a restrictive web search interface. Input capabilities provided by such aweb interface range from a simple keyword-search textbox (e.g., Google) to a complex combination of textboxes, drop-down 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 usu-ally a small integer such as 50 or 100. In fact, many web hid-den databases deliver their top-k results for a query with several web pages.

Unlike static webpages (connected by hyperlinks), the contents of a hidden database cannot be easily crawled by traditional web search engines, or by any method at all [1]. In fact, the restrictive web interface prevents users from per-forming complete queries as they would with the SQL lan-guage. For example, there are hardly any web interfaces providing aggregate queries such as COUNT and SUM functions. The lower 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 han-dle more complex queries as defined in standard SQL (i.e., aggregate functions) solely by issuing search queries through its restrictive web interface. In fact, such aggregate queries are desired by many applications which take hidden databases as their data sources.

We 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 websites, has an interface that features 95 checkbox attrib-utes. A Food search website [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, we 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 web 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 web interface. For example, in the home search website [4], features (e.g., central air, basement) for a home are represented

by check-boxes. 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 hid-den database, all existing techniques rely on (an often) unre-alistic 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 t½A& ¼6 v from the returned result.

The limitation placed by the checkbox interface prevents the traditional hidden-database aggregate-estimation tech-niques 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 web interface.

It is important to note that, in addition to the checkbox-attribute-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 web interface.

1.2 Outline of Technical Results

In this paper, we develop three main ideas for aggregate esti-mation over the hidden databases with checkbox interfaces:

UNBIASED-EST. We start by showing a unique chal-lenge imposed by the hidden databases with checkbox interfaces. Note that a common theme of the existing ana-lytic techniques for hidden web 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 esti-mation can be made) through sampling the query set. For the hidden databases with checkbox interfaces, however, 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 num-bers of queries (and therefore retrieved with different prob-abilities). 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, we are capable of mapping each tuple in the hidden database to exactly one query in the tree, which we 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 desig-nated one for its returned tuple(s), we develop an aggregate estimation algorithm that provides completely unbiased estimates for COUNT and SUM queries.

WEIGHTED-EST. The error of an aggregate estimation consists of two components: bias and variance.[1] Given that our first idea guarantees zero bias, we develop our second idea of weighted sampling to minimize variance. Specifically, we dynamically adjust the probability of sampling a query based on the query answers we receive so far, in order to

1. Specifically, the mean square error (MSE) of aggregate estimation satisfies $MSE = bias^2 + variance$.

“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, we 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 pred-icates—can cause a significant increase in the variance of aggregation estimations. To address the problem, we develop a special-case handling procedure which crawls such tuples to significantly reduce our final estimation error.

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

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

We introduce a novel problem of aggregate estima-tions over the hidden web databases with checkbox inter-faces, and outline the unique challenges it presents, which prevent the traditional hidden-database-sam-pling techniques from being applied.

To produce unbiased aggregate estimations over the hidden databases with checkbox interfaces, we 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 web interface.

To reduce the variance of aggregate estimations, we develop the ideas of weighted sampling and special-tuple-crawling.

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

1.3 Organization of the Rest of the Paper

The rest of the paper is organized as follows: In Section 2, we introduce the preliminaries. Then we start by focusing on the COUNT estimation problem and introduce our unbiased estimation algorithm in Section 3. Weight adjustment and low probability crawl will be introduced in Section 4. Experi-mental results are shown in Section 5. Related works and conclusion are produced in Sections 6 and 7 respectively.

2 PRELIMINARIES

2.1 Model of Hidden Databases with Checkboxes

In most parts of the paper, we focus on the case where a hid-den 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 attribute A_i 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 specify-ing values of a subset of attributes. Now suppose a user selects checkboxes A_{i1} ; . . . ; A_{ij} from the interface. With such selec-tions, 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} g$_q$ or directly fA_{i1} ; . . . ; A_{ij} g$_q$ in the later part of the paper for the sake of simplicity. Notation fg$_q$ 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 fjSelðq Þj; 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:

jSelðqÞj ¼ 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 cor-responding 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 over-flow flag.

2.3 Problem Definition

Any query of a hidden database with a checkbox interface can be represented into a SQL statement as:. ; 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 defini-tion of the problem is as follows.

Given a query budget G and an aggregate query Q: SELECT AGGRð Þ FROM D WHERE A_{i1} ¼ V_{i1} AND
AND A_{ij} ¼ V_{ij} , where AGGRð Þ is COUNT, SUM or AVG, and V_{i1} ; . . . ; V_{ij} 2 f0; 1g are values specified for checkboxes,
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 gath-ered 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.

3.1 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 sam-ple 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.

Recall that D has m checkbox attributes as its query inter-face, 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 possi-ble 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 orga-nize all these queries in the query space with a left-deep tree structure as shown in Fig. 1, where every node is corre-sponding 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.

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 esti-mator u used to estimate an aggregate query with real answer u, the relative error of u is defined as

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 aggre-gate 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 rela-tive error using less query cost.

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 data-base sampling.

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. But the

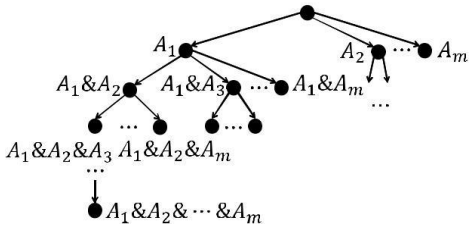


Fig. 1. A left deep tree.

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 map-ping between queries and tuples such that we can derive the probability for a tuple to be sampled from the proba-bility we used to sample the query which returns the tuple. We assign a tuple to exactly one query, called desig-nated 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 jdðq Þj 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 addi-tional 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 $fA_{i1} \& A_{i2} \&$ $\& A_{ih}\ g_q$ 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:

where function alphabeticðstringÞ outputs string in alpha-betic order. Thus, under this definition, any query is trans-formed 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 suc-ceeds q_1.

The above definition gives a complete order over the query set of our hidden database. For example, queries q_1 ¼ fA_1g_q, q_2 ¼ fA_2g_q, and q_3 ¼ fA_1 & A_2g_q, are ordered as q_1 q_3 q_2. With this definition, we can define a rule to solve the problem caused by multiply returned tuples in the hidden database.

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 actu-ally perform such heavy testing. Rather, only one additional query testing is necessary.

> 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 desig-nated 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 condi-tions should be satisfied. 1). For any attribute A_i, t½A_i& ¼ 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. ut

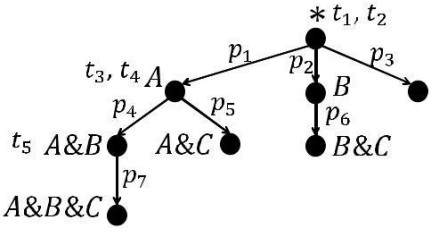
So, for each query, we need one additional query for des-ignated query testing of returned tuples.

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. 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 num-ber 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

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 assump-tion rarely fits in practice. As a result, UNBIASED estima-tion 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 prob-ability distribution. We shall propose an automatic weight adjustment algorithm to significantly reduce the estimation variance in next section.

4.1 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 corre-lated with each other. Take the hidden database Car Finder as an example, if a car contains leather seats, it usually con-tains 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 corre-lations among attributes may exist. Therefore, we cannot simply decompose Equation (9) into individual attribute distributions. Rather, it should be computed with consider-ation of the correlations between attributes. To compute the

With m attributes, there are as many as 2^m joint probabil-ities 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 signifi-cant 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.

5 Experiment

In this section, we describe our experimental setup and con-duct 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 con-ducted 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 algo-rithms 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 inde-

pendent and identical distribution of attributes. Let A_i, i ¼ 1 . . . m are attributes of the data set, pðA_iÞ is the probabil-ity for A_i ¼ 1. Here we set pðA_iÞ ¼ 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_1Þ ¼ pðA_2 Þ ¼ 0:1. For A_3 to A_m, we set pðA_3Þ ¼ 1=90; pðA_4Þ ¼ 2=90; . . . ; pðA_mÞ ¼ m=90 with a step of 1=90, where m is the number of attributes.

3) Aggregate estimation algorithms. We tested the four pro-posed 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 incremen-tally adjusts the drill-down weights after w queries. The weight assignment follows an assumption that the attrib-utes are independent.

UNBIASED-WEIGHTED. This algorithm assumes attrib-utes 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.

6 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 approxi-mation processing of aggregate queries, we develop algorithm UNBIASED-WEIGHTED-CRAWL which per-forms 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 vary-ing 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 analy-sis, 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 data-bases," 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 optimi-zation 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 limi-tations," 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-SIGACT-SIGART 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

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

[16] R. J. Lipton and J. F. Naughton, "Query size estimation by adap-tive 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 selec-tivity estimation through adaptive sampling," Sigmod Rec., vol. 19, PP. 1–11, 1990.