

Attribute Domain Discovery for Hidden Web Databases

Xin Jin
George Washington University
Washington, DC 20052, USA
xjin@gwu.edu

Nan Zhang^{*}
George Washington University
Washington, DC 20052, USA
nzhang10@gwu.edu

Gautam Das[†]
University of Texas at Arlington
Arlington, TX 76019, USA
gdas@uta.edu

ABSTRACT

Many web databases are hidden behind restrictive form-like interfaces which may or may not provide domain information for an attribute. When attribute domains are not available, domain discovery becomes a critical challenge facing the application of a broad range of existing techniques on third-party analytical and mash-up applications over hidden databases. In this paper, we consider the problem of domain discovery over a hidden database through its web interface. We prove that for any database schema, an achievability guarantee on domain discovery can be made based solely upon the interface design. We also develop novel techniques which provide effective guarantees on the comprehensiveness of domain discovery. 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 Web Database, Domain Discovery

1. INTRODUCTION

The Attribute Domain Discovery Problem: In this paper, we develop novel techniques to discover the attribute domains, i.e., the set of possible values for each attribute, from hidden web databases, by external users. Hidden databases, as a large portion of the deep

^{*}Partially supported by NSF grants 0852673, 0852674, 0915834 and a GWU Research Enhancement Fund.

[†]Partially supported by NSF grants 0812601, 0915834, 1018865, a NHARP grant from the Texas Higher Education Coordinating Board, 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'11, June 12–16, 2011, Athens, Greece.

Copyright 2011 ACM 978-1-4503-0661-4/11/06 ...\$10.00.

web, are hidden behind restrictive form-like interfaces which allow a user to form a search query by specifying the desired values for one or a few attributes; and the system responds by returning a small number of tuples satisfying the user-specified search condition. A typical example of a hidden database is the award search database of the US National Science Foundation (NSF)¹, which allows users to search for NSF-award projects featuring user-specified values on up to 20 attributes.

Each attribute specifiable by users through the form-like interface is represented by an input control on the interface. For certain types of controls, e.g., drop-down menus and radio buttons, the attribute domain can be readily retrieved by external users from source code of the interface (e.g., an HTML file). In the NSF example, attributes such as NSF organization and Award Instrument belong to this category. For other controls, especially text-boxes (without features such as autocompletion), no domain information is provided. Attributes such as Program Manager belong to this category.

The focus of this paper is to develop domain discovery techniques, *restricted to accessing the database only via the proprietary form interface*, that external users can employ to unveil possible attribute values that appear in tuples of the hidden databases. An important design goal is to develop techniques that unveil all (or most) attribute values by issuing only a small number of queries, and to provide analytical guarantees on the coverage and query cost of our methods.

We emphasize that in this paper, we do not consider approaches to the domain discovery problem that rely on external knowledge sources or domain experts to provide the required domain values. We argue that while the use of external knowledge sources may have applicability in very general scenarios (e.g., the use of a USPS data source to list all values of an attribute such as zipcode), these approaches will not work in more focused applications such as the Program Manager attribute on the NSF award search database. Our emphasis is to develop automated domain discovery algorithms that can power a variety of third-party applications only using the public interfaces of these databases, and without requiring any further collaborations or agreements with the database owners.

Applications: Domain discovery from hidden databases belongs to the area of information extraction and deep web analytics (see tutorials in [6, 11] and survey in [18]). To the best of our knowledge, the only prior work that tackles our specific problem of discovering attribute domains of hidden databases is [20]. This problem has a broad range of applications. First and foremost, it serves as a *critical prerequisite* for data analytics over hidden databases, because all existing aggregate estimation [8] and sampling [7, 9, 10] techniques for hidden databases require prior knowledge of the attribute

¹<http://www.nsf.gov/awardsearch/tab.do?dispatch=4>

domains. In addition, the attribute domains being discovered can be of direct interest to third-party web mashup applications. For example, such an application may use the discovered attribute domains to add autocomplete feature to text-boxes in the original form-like interface, so as to improve the usability of the interface. The discovered domains may also be used to identify mapping between attributes in different hidden databases to facilitate the creation of mashup.

Challenges: The main challenges to effective attribute domain discovery are the restrictions on input and output interfaces of hidden databases. Normally, input interfaces are restricted to issuing only *search* queries with conjunctive selection conditions - which means that queries like `SELECT UNIQUE(Program Manager) FROM D` cannot be directly issued, eliminating the chance of directly discovering the domain of `Program Manager` through the interface.

The output interface is usually limited by a top- k constraint - i.e., if more than k tuples match the user-specified condition, then only k of them are preferentially selected by a scoring function and returned to the user. This restriction eliminates the possibility of trivially solving the problem by issuing the single query `SELECT * FROM D` and then discovering all attribute domains from the returned results.

The existing technique for attribute domain discovery is based on *crawling* the database [20]. A simple instantiation of crawling is to start with `SELECT * FROM D`, then use domain values returned in the query answer to construct future queries, and repeat this process until all queries that can be constructed have been issued. One can see that no algorithm can beat the comprehensiveness of attribute domain discovery achieved by crawling. But crawling requires an extremely large number of queries to complete, and since most hidden databases enforce a limit on the number of queries one can issue over a period of time (e.g., 1,000 per IP address per day), or even charge per query, often the crawling process has to be terminated prematurely. Due to this reason, such techniques cannot provide any guarantees on the comprehensiveness of discovered attribute domains.

A seemingly promising way to address the query cost problem of crawling is to instead perform *sampling* - i.e., one first draws representative samples from the hidden database, and then discover attribute domains from the sample tuples to achieve statistical guarantees on the comprehensiveness of domain discovery. Sampling of hidden databases is an intensively studied problem in recent years [7, 9, 10]. Nonetheless, a key obstacle here is a “chicken-and-egg” problem - i.e., all existing sampling algorithms for hidden databases require the attribute domains to be known in the first place, preventing them from being applied to hidden databases with unknown attribute domains!

Outline of Technical Results: In this paper we initiate a study of query-efficient attribute domain discovery over hidden databases. We consider two types of output interfaces commonly offered by hidden databases: `COUNT` and `ALERT`. For a user-specified query, a `COUNT` interface returns not only the top- k tuples, but also the number of tuples matching the query in the database (which can be greater than k). `ALERT` interface, on the other hand, only *alerts* the user with an *overflow flag* when more than k tuples match the query, indicating that not all tuples which match the query can be returned. The NSF award database features an `ALERT` interface.

For `COUNT` interface, we start by studying the feasibility of developing deterministic algorithms for attribute domain discovery. Our main results are two achievability conditions - lower bounds on k and query cost, respectively - which a deterministic algorithm must satisfy to guarantee the discovery of all attribute domains. Unfortunately, neither condition is achievable in practice for a generic

hidden database. Nonetheless, a promising lead we find from the study is that efficient and complete attribute domain discovery is indeed possible for a special type of (perhaps unrealistic) databases in which each attributes has only two domain values. For this special case, we develop `B-COUNT-DISCOVER`, a deterministic algorithm based on the key idea of constructing a series of *nested search spaces* for unknown domain values, and prove that the algorithm guarantees the discovery of all attribute domains with only $O(m^2)$ queries, where m is the number of attributes, as long as $k \geq 2$.

Our main observation from `B-COUNT-DISCOVER` is an understanding of why it cannot be extended to generic hidden databases containing attributes with arbitrary-sized domains - we find the main reason to be the large number of queries required by a deterministic algorithm to decide how to construct the nested search space. To address this, we consider the design of randomized algorithms which allows query-efficient randomization of the nested-space construction process. To this end, we develop `RANDOM-COUNT-DISCOVER`, a Monte Carlo algorithm which only requires a small amount of queries to discover a large number of domain values.

For `ALERT` interface, we similarly prove the infeasibility of developing a query-efficient deterministic algorithm which guarantees complete attribute domain discovery. Our main result here is then an extension of `RANDOM-COUNT-DISCOVER` to `RANDOM-ALERT-DISCOVER`, a Monte Carlo algorithm which achieves similar performance to `RANDOM-COUNT-DISCOVER` by estimating `COUNTs` of queries based on historic query answers, and using estimated `COUNTs` to bootstrap `RANDOM-COUNT-DISCOVER` to discover domain values.

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

- We initiate the study of attribute domain discovery over a hidden database through its restrictive web interface.
- We derive the achievability conditions for complete attribute domain discovery on k and the query cost. These conditions indicate the infeasibility of designing a query-efficient deterministic algorithm which guarantees the discovery of all attribute domains.
- We propose two randomized algorithms, `RANDOM-COUNT-DISCOVER` and `RANDOM-ALERT-DISCOVER`, for `COUNT` and `ALERT` interfaces, respectively. Both algorithms require only a small amount of queries to develop a large number of domain values.
- We provide a thorough theoretical analysis and experimental studies that demonstrate the effectiveness of our proposed algorithms over the real-world NSF award search database and a local patient discharge dataset.

Paper Organization: In Section 2 we introduce preliminaries and discuss simple but ineffective crawling algorithms for attribute domain discovery over hidden databases. Sections 3 and 4 are devoted to the development of deterministic and randomized algorithms for `COUNT` interfaces, respectively. In Section 5 we extend the results to `ALERT` interfaces. 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 Model of Hidden Databases

Consider a hidden database table D with n tuples. Let there be m attributes A_1, \dots, A_m which can be specified through the input interface, and V_i be the domain of A_i . We assume each tuple to be unique, and each value in V_i ($i \in [1, m]$) to occur in at least one tuple in the database, because otherwise V_i can never be completely discovered from the database. Consider a prototypical interface which allows user to query D by specifying values for a subset of attributes - i.e., to issue queries q of the form:

```
SELECT * FROM D WHERE  $A_{i_1} = v_{i_1} \& \dots \& A_{i_s} = v_{i_s}$ ,
```

where $v_{i_j} \in V_{i_j}$. Let $Sel(q)$ be the set of tuples in D which satisfy q . Since the interface is restricted to return up to k tuples, a overly broad query (i.e., $|Sel(q)| > k$) will *overflow* and return only the top- k tuples in $Sel(q)$ selected according to a scoring function. In addition, a COUNT interface will return $|Sel(q)|$, the number of tuples satisfying q , while an ALERT interface will return an overflow flag indicating that not all tuples which satisfy q can be returned.

When a query does not overflow, COUNT and ALERT interfaces return the exact same information to the user. In particular, if the query is too specific to return any tuple, we say that an *underflow* occurs. If there is neither overflow nor underflow (i.e., $|Sel(q)| \in [1, k]$), then $Sel(q)$ will be returned in its entirety and we have a *valid* query result. For each tuple returned by an overflowing or valid query, the values of all its attributes are displayed, enabling the discovery of domain values of an attribute with unknown domain.

Running Example: We consider a running example of the above-mentioned NSF award search database with $k = 50$. There are 9 attributes, award amount, instrument, PI state, field, program manager, NSF organization, PI organization, City, and ZIP code, with domain sizes 5, 8, 58, 49, 654, 58, 3110, 1093, 1995, respectively (these are the estimates we generated by running our domain discovery algorithms to be presented in the paper for an extended period of time. We do not have the ground truth for all attributes. But they can be safely considered as lower bounds on domain sizes).

2.2 Problem Definition

We consider the problem of attribute domain discovery - i.e., the discovery of V_1, \dots, V_m through the restrictive interface. Since many hidden databases impose limits on the number of queries one can issue through their interfaces over a period of time, the performance of a domain discovery algorithm should be measured by not only the *comprehensiveness* of discovered domain values, but also the *query cost*, i.e., the number of queries issued through the interface of hidden database for domain discovery.

While the metric for query cost is straightforward, to measure the comprehensiveness of discovered domain values we consider the following two metrics.

- *Coverage*, i.e., the total number of domain values discovered for one or a set of attributes.
- *Recall*, i.e., the number of tuples in the database for which all attribute values (of A_1, \dots, A_m) have been discovered.

Note that while the two metrics are positively correlated (e.g., a recall of n indicates a coverage of $|V_i|$ for each A_i), they may be preferred in different applications. For example, if the objective of domain discovery is to unveil the metadata, e.g., to discover the list of program managers from the NSF award database, then coverage is a more important metric. If the objective is to enable data analytics over the hidden database, then recall is more important as

it guarantees how many tuples will be covered by the subsequent aggregate estimation and sampling algorithms.

Given the measures for comprehensiveness and query cost, we define the problem of attribute domain discovery as follows:

Problem Statement: *How to maximize the coverage and recall of attribute domain discovery while minimizing the query cost through COUNT and ALERT interfaces, respectively.*

2.3 Crawling-Based Algorithms

In this subsection, we consider depth-first search (DFS) and breadth-first search (BFS), two simple crawling-based algorithm for attribute domain discovery, and point out their problems in terms of the tradeoff between coverage/recall and query cost. Both algorithms start with SELECT * FROM D to learn the first few domain values, but then take different methods to construct the subsequent queries.

DFS: With DFS, one constructs the next query by finding an attribute that is not specified in the previous one, and then adding to the previous query a conjunctive condition defined by the attribute and one of its already discovered domain values. Such a process continues, with DFS learning all domain values from tuples returned by each issued query, until either the query returns underflows/valid or no discovered domain value is available for constructing the next query, at which time the algorithm backtracks by removing the last-added condition and adding a new one based on another discovered domain value. We refer to the algorithm as DFS because it keeps increasing the number of predicates included in the query if possible.

A main problem of DFS, however, is poor coverage and recall when a large number of attributes are strongly correlated with the first few attributes appended to SELECT * FROM D. For example, consider the case where $A_1 = v_1$ is first appended to the SELECT * query, and $A_1 \rightarrow A_m$ forms a functional dependency. Since DFS with a small query budget is likely to explore only queries with predicate $A_1 = v_1$, the domain values of A_m which are corresponding to the other values of A_1 will not be discovered.

BFS: With BFS, one first exhausts all 1-predicate queries that can be constructed from discovered domain values, before moving forward to construct and issue 2-predicate queries, etc. The main problem of BFS arises when a large number of domain values are strongly correlated with the scoring function used to select the top- k tuples, because BFS with a small query budget is likely to issue only overflowing queries. For example, consider a total order of values for A_m and suppose that tuples with “larger” A_m receives higher scores. Then, BFS will likely to only discover the larger values of A_m .

3. DETERMINISTIC ALGORITHMS FOR COUNT INTERFACES

In this section, we analyze the achievability conditions for attribute domain discovery over a top- k -COUNT interface, and also develop deterministic algorithms which guarantee the discovery of all attribute domains when the achievability conditions are met. The purpose of introducing deterministic algorithms is to use them as the basis for our design of more efficient randomized algorithms in the next section.

3.1 Discover Binary Domains

We start with a simple case where each attribute has only two possible values, both unknown to the third-party analyzer². An ex-

²The third-party analyzer may or may not know the binary nature of all attributes - our results apply either way.

ample of such an attribute is **transmission** for an auto database which has two values, “manual” and “automatic”. While hardly any practical database consists solely of binary attributes, the results in this subsection are illustrative when considering extensions to handle arbitrary attribute domains, as we shall show in the next subsection.

3.1.1 Achievability of Domain Discovery

Before devising concrete attribute domain discovery algorithms, we first consider the problem of achievability - i.e., whether it is at all possible for the third-party analyzer to discover all attribute domains from the top- k interface. There are two sub-problems: (1) whether complete attribute domain discovery is possible when the analyzer can afford an unlimited query cost, and (2) if the first answer is yes, then what is the minimum number of queries required by a deterministic algorithm to accomplish attribute domain discovery in the worst case?

Our main results, as shown by the following three theorems, can be summarized as follows: The answer to the first question is a lower bound on k - i.e., no algorithm without knowledge of the scoring function can guarantee complete discovery over any hidden database if $k = 1$, while complete discovery is always possible for any combination of database and scoring function when $k \geq 2$. For the second question, if $k \geq 2$, a deterministic algorithm require (at least) $\Omega(m^2 / \log m)$ queries to discover all attribute domains in the worst case, where m is the number of attributes, unless $k \geq n$, in which case **SELECT * FROM D** returns all tuples in the database.

THEOREM 3.1. *If $k = 1$, then for any given binary database, there exists a scoring function such that no algorithm can discover the domains of all attributes.*

PROOF. This impossibility proof is simple - Let the tuple returned by **SELECT * FROM D** be $t = [0_1, \dots, 0_n]$. Assign the highest score to t . One can see that no algorithm can discover any domain value other than $\{0_1, \dots, 0_n\}$. \square

THEOREM 3.2. *For any given number of attributes m , there exists a binary database and a scoring function such that no deterministic algorithm can complete attribute domain discovery without incurring a worst-case query cost of $\Omega(m^2 / \log m)$, even if $k \geq 2$.*

PROOF. For the sake of simplicity, we consider the case where m is even. The case where m is odd can be proved in analogy. We construct a set of $2^{m^2/4}$ database instances $D_1, \dots, D_{2^{m^2/4}}$ as follows: Each instance D_i consists of $2^{m/2} + m/2$ tuples $t_1, \dots, t_{2^{m/2} + m/2}$.

We consider the first $2^{m/2}$ tuples first - for these tuples (i.e., with $i \in [1, 2^{m/2}]$), if $j \in [1, m/2]$, $t_i[A_j] = 0_j$ (resp. 1_j) iff the j -th most significant bit of the binary representation of $i - 1$ is 0 (resp. 1); if $j \in [m/2 + 1, m]$, then there is always $t_i[A_j] = 0_j$.

For the latter $m/2$ tuples, their values of $A_{m/2+1}, \dots, A_m$ are set such that $t_i[A_j] = 1_j$ iff $i = 2^{m/2} - m/2 + j$, and $t_i[A_j] = 0_j$ otherwise. For the values of $A_1, \dots, A_{m/2}$ of these tuples, they are set such that each database instance has a different combination of the $m/2$ tuples. Note that for each tuple, there are $2^{m/2}$ possible choices. Thus, we can construct $(2^{m/2})^{m/2} = 2^{m^2/4}$ different database instances. The scoring function for each instance is designed with only one condition: each tuple which satisfies $t_i[A_j] = 0_j$ for all $j \in [m/2 + 1, m]$ has a higher score than all tuples which do not satisfy this condition.

There are two important observations one can make from the above construction when $k = 2$. First, each database instance requires a different sequence of queries to unveil all domain values. The reason for this is because the only way to unveil 1_j for

$j \in [m/2 + 1, m]$ is to issue a query that contains at least $m/2$ predicates corresponding to the values of $A_1, \dots, A_{m/2}$ for tuple $t_{2^{m/2} - m/2 + j}$, respectively. Since each database instance has a different combination of values for $t_{2^{m/2} + 1}, \dots, t_{2^{m/2} + m/2}$, each instance requires a different sequence of queries for complete attribute domain discovery.

The second important observation is that the ability for any query to distinguish between the $2^{m^2/4}$ database instances is limited. Note that if only the returned tuples are concerned, then almost all queries (except those that unveil one of 1_j for $j \in [m/2 + 1, m]$) return the exact same tuples when $k = 2$. The more powerful distinguishing factor is COUNT. Nonetheless, the COUNT of any query has only $m/2$ different answers for all database instances, because after all these instances differ by at most $m/2$ tuples.

Since we focus on deterministic algorithms, it is impossible for an algorithm to receive the same answer for all previous queries but then issue a different query in the next step for two database instances. Thus, in order for the algorithm to have a different sequence of queries for each database instance, the query cost is at least $\log_{m/2} 2^{m^2/4}$, i.e., $\Omega(m^2 / \log m)$. \square

Given the achievability conditions shown in the above two theorems, we now show the existence of a deterministic algorithm that is capable of discovering all attribute domains when the achievability conditions are met, and achieves a near-optimal query cost (within a factor of $\log m$ where m is the number of attributes).

THEOREM 3.3. *If $k \geq 2$, there exists a deterministic algorithm which guarantees complete attribute domain discovery for all binary databases and all scoring functions with a query cost of $O(m^2)$.*

The following deterministic algorithm, **B-COUNT-DISCOVER**, serves as the proof. One can see from this theorem that for databases with only binary domains, the problem of attribute domain discovery can indeed be solved with a small query cost as long as $k \geq 2$.

3.1.2 B-COUNT-DISCOVER

B-COUNT-DISCOVER starts by issuing query q_1 : **SELECT * FROM D**. Without loss of generality, let $t : [0_1, \dots, 0_m]$ be a tuple returned by q_1 . Since $k \geq 2$, there must be another tuple returned by q_1 which differs from t on at least one attribute value. Again without loss of generality, let such an attribute be A_1 - i.e., the analyzer learns from the answer to q_1 at least the following $m + 1$ values: $\{0_1, \dots, 0_m, 1_1\}$. The objective of **B-COUNT-DISCOVER** then becomes to unveil $1_2, \dots, 1_m$.

To discover any of these values, say 1_m , we essentially need to find a tuple with $A_m = 1_m$. Initially, the *search space* - i.e., the set of possible tuple values in the database featuring $A_m = 1_m$ - is very large. In particular, the initial space $S_1(1_m)$ is formed by all possible value combinations for A_1, \dots, A_{m-1} (and $A_m = 1_m$) and thus of size 2^{m-1} . Our main idea of **B-COUNT-DISCOVER** is to shrink the search space by leveraging COUNT information provided by the interface. In particular, we construct a series of *nested spaces* $S_1(1_m), S_2(1_m), \dots$, each half the size of the preceding one, while ensuring that every $S_i(1_m)$ contains at least one tuple in the database with $A_m = 1_m$. Then, even in the worst-case scenario, we can unveil 1_m when the search space is shrunk to size 1 - by simply issuing an $(m-1)$ -predicate query with A_1, \dots, A_{m-1} specified according to the value left in the search space.

To understand how the nested spaces are constructed, consider the following three queries.

q_2 . **SELECT * FROM D WHERE $A_m = 0_m$**

q_3 . **SELECT * FROM D WHERE $A_1 = 0_1$**

q_4 . **SELECT * FROM D WHERE $A_1 = 0_1$ AND $A_m = 0_m$**

Let $C(q_i)$ be the COUNT returned alongside query q_i . We can compute from the query answers $C_1 : \text{COUNT}(A_1 = 0_1 \text{ AND } A_m = 1_m)$ and $C_2 : \text{COUNT}(A_1 = 1_1 \text{ AND } A_m = 1_m)$ as

$$C_1 = C(q_3) - C(q_4), \quad (1)$$

$$C_2 = C(q_1) - C(q_2) - (C(q_3) - C(q_4)). \quad (2)$$

Note that either C_1 or C_2 (or both) must be greater than 0 if 1_m occurs in the database. Suppose that $C_1 > 0$. We can now shrink our search space by half to size 2^{m-2} - by constructing $\mathcal{S}_2(1_m) \subset \mathcal{S}_1(1_m)$ with only values in $\mathcal{S}_1(1_m)$ which satisfy $A_1 = 0_1$. One can see that, since $C_1 > 0$, at least one value in $\mathcal{S}_2(1_m)$ appears in the database and has $A_m = 1_m$.

The shrinking process continues from here. To see how, note that since $C_1 > 0$, q_3 either returns a tuple with $A_m = 1_m$ or overflows. If q_3 overflows without returning 1_m , there must exist another attribute (in A_2, \dots, A_{m-1}) which has both values appearing in the k tuples returned for q_3 . Without loss of generality, let the attribute be A_2 . We now issue the following two queries.

q_5 . SELECT * FROM D WHERE $A_1 = 0_1$ AND $A_2 = 0_2$
 q_6 . SELECT * FROM D WHERE $A_1 = 0_1$ AND $A_2 = 0_2$ AND $A_m = 0_m$

Similar to the last step, we compute $C_3 : \text{COUNT}(A_1 = 0_1 \text{ AND } A_2 = 0_2 \text{ AND } A_m = 1_m)$ and $C_4 : \text{COUNT}(A_1 = 0_1 \text{ AND } A_2 = 1_2 \text{ AND } A_m = 1_m)$ as

$$C_3 = C(q_5) - C(q_6), \quad (3)$$

$$C_4 = C(q_3) - C(q_4) - (C(q_5) - C(q_6)). \quad (4)$$

Again, at least one of C_3 and C_4 must be greater than 0 because $C_1 > 0$. Suppose that $C_4 > 0$. We can then further shrink our search space to size 2^{m-3} by constructing $\mathcal{S}_3(1_m) \subset \mathcal{S}_2(1_m)$ with only values in $\mathcal{S}_2(1_m)$ which satisfies $A_2 = 1_2$. Once again, at least one value in $\mathcal{S}_3(1_m)$ appears in the database and has $A_m = 1_m$ because $C_4 > 0$. Note that before the next shrinking step, we might need to issue

q_7 . SELECT * FROM D WHERE $A_1 = 0_1$ AND $A_2 = 1_2$
in order to discover the complete domain of another attribute (in A_3, \dots, A_{m-1}), so that we can continue the shrinking process. Note that similar to the discovery of A_2 's domain from q_3 , this discovery is guaranteed by q_7 because it either returns a tuple with $A_m = 1_m$ (which accomplishes our task) or overflows, in which case at least one attribute unspecified in q_7 must have different values appear in the $k \geq 2$ tuples returned by q_7 .

One can see that eventually, we can always discover 1_m after issuing at most $3m$ queries, because the search space would then become $\mathcal{S}_m(1_m)$ of size 1. Thus, B-COUNT-DISCOVER requires a query cost of $O(m^2)$. This concludes the proof of Theorem 3.3.

3.2 Discover Arbitrary Domains

We now extend our results for binary domains to arbitrary domains. We start by showing that the originally practical achievability conditions for binary domains, i.e., lower bounds on k and query cost, now become unrealistic for arbitrary domains. Then, we illustrate the fundamental reason for such a change which motivates our design of randomized algorithms in the next section.

3.2.1 Achievability of Attribute Domain Discovery

For arbitrary domains, the achievability condition for attribute domain discovery imposes a much large lower bound on k , as indicated by the following theorem. Recall that $|V_i|$ is the domain size of A_i .

THEOREM 3.4. *If $k < 1 + \prod_{i=1}^m (|V_i| - 1)$, then no deterministic algorithm can guarantee the discovery of all attribute domains for all database instance/scoring function combinations.*

PROOF. Arbitrarily pick one attribute value v_i from each V_i ($i \in [1, m]$), respectively. Consider a database D formed by $1 + \prod_{i=1}^m (|V_i| - 1)$ tuples. $\prod_{i=1}^m (|V_i| - 1)$ of them are the Cartesian product of $V_i \setminus \{v_i\}$ for all $i \in [1, m]$. The one additional tuple is $t : [v_1, \dots, v_m]$. Suppose that t has the lowest score. One can see that no algorithm can discover any of v_1, \dots, v_m if t is not returned by SELECT * FROM D. Thus, no algorithm can guarantee complete attribute domain discovery when $k < 1 + \prod_{i=1}^m (|V_i| - 1)$. \square

One can see from Theorem 3.4 that the binary-case result, Theorem 3.1, is indeed a special case when $|V_1| = \dots = |V_m| = 2$. Another observation from Theorem 3.4 is that the lower bound on k has now become hardly reachable because real-world database often features a number of attributes with large domains.

Running Example: The NSF award search database requires $k \geq 1 + \prod_{i=1}^m (|V_i| - 1) = 1.93 \times 10^{19}$ to guarantee complete attribute domain discovery.

What reinforces this impracticability result is the following theorem. Note that since the achievability condition on k is now infeasible, it makes little practical sense for us to assume $k \geq 1 + \prod_{i=1}^m (|V_i| - 1)$, as in the binary case, when deriving the lower bound on query cost. As such, we consider the following question: for a given value of k , is it possible for a deterministic algorithm to use a small number of queries to discover all domain values that can be discovered from such a top- k interface (given unlimited query cost)? The following lower bound on k shows that the answer to this question is also no for hidden databases with arbitrary domains.

THEOREM 3.5. *For given k and m , there exists a database and a scoring function such that no deterministic algorithm can complete the discovery of all domain values that can be discovered from the top- k COUNT interface without incurring a worst-case query cost of*

$$\Omega \left(\frac{m^2 \cdot |V_{\max}| \cdot \log |V_{\max}|}{k \cdot \log(m \cdot |V_{\min}|)} \right) \quad (5)$$

where $|V_{\max}|$ and $|V_{\min}|$ are the maximum and minimum values in $|V_1|, \dots, |V_m|$, respectively.

The proof of this theorem follows in analogy to Theorem 3.2. We do not include it due to the space limitation. One can see that the binary-case lower bound on query cost, Theorem 3.2, is indeed a special case of Theorem 3.5 when $|V| = 2$ and $k = 2$. Another observation from Theorem 3.5 is that this arbitrary-domain lower bound on query cost, just like the lower bound on k , is hardly achievable in practice, especially when the hidden database contains one or a few attributes (e.g., ZIP code) with large domains.

Running Example: With the setting of our NSF award search example, $m^2 \cdot |V_{\max}| \cdot \log |V_{\max}| / (k \cdot \log(m \cdot |V_{\min}|)) = 1.06 \times 10^4$. While the bound may differ by a constant factor, 10^4 is at least an order of magnitude larger than what one would desire for attribute domain discovery (e.g., in our experimental results).

Although the achievability conditions for both k and query cost are unrealistic for arbitrary domains, in the following we still extend B-COUNT-DISCOVER to COUNT-DISCOVER, a deterministic algorithm that is capable of discovering all attribute domains. Of course, the extended algorithm has to follow the achievability conditions and therefore cannot provide meaningful worst-case performance. Our main purpose of studying it is to identify which part of the extension causes the significant increase on query cost. Theorem 3.6 summarizes the query cost of COUNT-DISCOVER.

THEOREM 3.6. *For an m -attribute database with $k \geq 1 + \prod_{i=1}^n (|V_i| - 1)$, there exists an algorithm which requires at most*

$$\sum_{i=1}^m \left((2|V_{d_i}| - 1) \cdot \sum_{j=1}^{m-i+1} \left(\frac{|V_{d_j}| \cdot (|V_{d_j}| - 1)}{2} \right) \right) \quad (6)$$

queries to discover all attribute domains, where d_1, \dots, d_m is the permutation of $1, \dots, m$ which satisfies $|V_{d_1}| \geq \dots \geq |V_{d_m}|$.

Running Example: With NSF award search example, the worst-case query cost of COUNT-DISCOVER is 1.07×10^{11} , orders of magnitude larger than what one can afford.

Similar to the binary case, we explain the proof of this theorem in the description of COUNT-DISCOVER. Note yet again that the binary-case result, Theorem 3.3, is a special case of Theorem 3.6 when $|V_i| = 2$ for all $i \in [1, m]$. Nonetheless, unlike in the binary case where B-COUNT-DISCOVER has a query cost within a factor of $O(\log m)$ from optimal, the query cost of COUNT-DISCOVER is further away from the lower bound in Theorem 3.5. In particular, observe from Theorem 3.6 the worst-case query cost of COUNT-DISCOVER is $\Omega(|V_{d_1}|^2 + |V_{d_m}|^3)$, a factor of $O(|V_{d_1}| + |V_{d_m}|^2)$ from optimal. We did not pursue to close this gap for two reasons: First, since the lower bound itself is infeasible in practice, even an optimal algorithm would not be practical anyway. Second, our purpose of introducing COUNT-DISCOVER is not to promote its practical usage, but to use the comparison between B-COUNT-DISCOVER and COUNT-DISCOVER to illustrate the main obstacle facing the discovery of arbitrary attribute domains, which motivates our design of efficient (and thus practical) randomized algorithms.

3.2.2 COUNT-DISCOVER

We now extend B-COUNT-DISCOVER to arbitrary domains, given the condition that $k \geq 1 + \prod_{i=1}^n (|V_i| - 1)$. Note that this lower bound on k guarantees that SELECT * FROM D must reveal the complete domain for at least one attribute, assumed to be A_1 without loss of generality. In addition, it must reveal at least one domain value for the remaining $m - 1$ attributes, assumed to be $0_2, \dots, 0_m$. As such, the objective of COUNT-DISCOVER is to unveil $1_i, \dots, (|V_i|)_i$ for all $i \in [2, m]$.

Like in the binary case, we consider the discovery of an arbitrary attribute domain, say V_m , by constructing a series of nested spaces that are guaranteed to contain at least one tuple in D which has its value of A_m yet to be discovered. Consider the worst-case scenario where the only value of V_m revealed by SELECT * FROM D is 0_m . The starting search space $\mathcal{S}_1(V_m)$ then has a size of $(|V_m| - 1) \cdot \prod_{i=1}^{m-1} |V_i|$. Since we already know the complete domain of A_1 , the next step is to find one value in V_1 which can be used to construct the next nested space - which must guarantee the appearance of at least one value in $1_m, \dots, (|V_i|)_m$.

In the binary case, we only need to determine whether 0_1 can be used because if it cannot, then 1_1 can always be used because A_1 has only these two possible values. In particular, the judgement for 0_1 can be done with just two queries - WHERE $A_1 = 0_1$ and WHERE $A_1 = 0_1$ and $A_m = 0_m$ - because we can then infer the COUNT of tuples that satisfy $A_1 = 0_1$ and $A_m = 1_m$. Note that for arbitrary domains, we can still determine whether 0_1 can be used with the two exact same queries, because they now reveal the COUNT of tuples that satisfy $A_1 = 0_1$ and have their values of A_m yet to be discovered. Nonetheless, since the domain of A_1 is much larger (than binary), we must continue testing other values of A_1 if 0_1 cannot be used. In the worst-case scenario, we may not be able to find the next search space until testing $|V_1| - 1$ possible

values of A_1 . Thus, at this step, the query cost for arbitrary domain becomes $|V_1| - 1$ times as large as that for binary domain.

Let $w_1 \in V_1$ be the value we find and use to construct the next nested search space. We can now construct the next nested space $\mathcal{S}_2(V_m)$ by only including value combinations in $\mathcal{S}_1(V_m)$ which satisfy $A_1 = w_1$. One can see that the size of search space now becomes $(|V_m| - 1) \cdot \prod_{i=2}^{m-1} |V_i|$. COUNT-DISCOVER continues the shrinking process in the same way as the binary case - i.e., by issuing query q : SELECT * FROM D WHERE $A_1 = w_1$ (if it has not already been issued). One can see that either q returns a yet-to-be-discovered value in V_m , or it overflows, in which case we can always find the complete domain of an attribute in A_2, \dots, A_{m-1} from the k tuples returned by q because $k \geq 1 + \prod_{i=1}^n (|V_i| - 1)$. As such, eventually we can always unveil one value in V_m that has not yet been discovered. Nonetheless, since we do not maintain in the search space *all* yet-to-be-discovered value in V_m , the entire shrinking process may have to be repeated multiple times to unveil the entire V_m . Theorem 3.6 can be derived accordingly.

A key observation from the design of COUNT-DISCOVER is that the significant increase on query cost (from the binary case) is not because the shrinking of search spaces becomes less powerful - indeed, ALERT-DISCOVER still needs only m nested spaces $\mathcal{S}_1(V_m), \dots, \mathcal{S}_m(V_m)$ to reach size 1. The increase on query cost is largely due to the cost of determining how to construct the next nested space (i.e. by filtering the current one) - a task previously accomplished by just two queries now requires many more. In the next section, we shall show that randomizing such nested-space construction process is our key idea for building efficient randomized algorithms for discovering arbitrary attribute domains.

4. RANDOMIZED ALGORITHMS FOR COUNT INTERFACES

In the last section, we developed a query-efficient algorithm for discovering binary domains, but found its extension to arbitrary domains to be extremely inefficient. We also found the main reason behind this obstacle to be the increased cost of determining how to construct the nested search spaces - in particular, for a given search space and an attribute A_i , how to select a domain value v of A_i such that $A_i = v$ can be used to filter the current search space while maintaining at least one tuple with yet-to-be-discovered domain value(s) in the filtered space.

Now that any deterministic construction of nested spaces is provably expensive due to the achievability conditions we derived in the last section, our main idea in this section is to *randomize* nested-space construction, so as to spend a small number of queries for search space construction while maintaining yet-to-be-discovered domain value(s) in the constructed space with high probability. In the following, we first describe our randomized process of nested-space construction and the performance (i.e., recall and coverage) guarantee it is able to achieve, and then combine this process with COUNT-DISCOVER to form RANDOM-COUNT-DISCOVER, our randomized (Monte Carlo) algorithm for attribute domain discovery which can unveil attribute domains (though not necessarily completely) no matter if the achievability conditions are met.

4.1 Randomize Nested-Space Construction

Recall that the main obstacle for nested-space construction is to find a predicate $A_i = v$ that can be used to filter the current search space. In the worst-case scenario, one may have to find $m - 1$ such conditions to discover a domain value for an attribute say A_j - this occurs when the complete domains of all attributes but A_j have been discovered. For the purpose of discussing the randomization of nested-space construction, we consider a worst-case scenario

where the complete domains of A_1, \dots, A_{m-1} have been discovered, and our objective now is to discover the domain of A_m . As a side note - this problem is actually interesting in its own right because it captures the scenario where A_m is a textbox attribute (with unknown domain) on the hidden database interface, while a large number of other attributes (i.e., A_1, \dots, A_{m-1}) have domains provided by the interface (e.g., through options in drop-down menus). We shall shown in the experiments section that our algorithms can effectively discover the domain of A_m in this scenario.

RANDOM-SPACE: A simple idea to randomize the construction of nested spaces is to choose v uniformly at random from the domain of A_i . With this idea, every round (i.e., search-space shrinking process) of our RANDOM-SPACE algorithm starts with `SELECT * FROM D` as the first query. Then, if no new domain value of A_m is discovered from the query answer, it randomly chooses v_1 from V_1 and then uses $A_1 = v_1$ to filter the search space (i.e., by issuing query `SELECT * FROM D WHERE A_1 = v_1`). More generally, if RANDOM-SPACE fails to discover a new domain value of A_m from the first $b-1$ queries issued in the current round, then it constructs the b -query by adding a conjunctive condition $A_b = v_b$, where v_b is chosen uniformly at random from V_b , to the selection condition of the $(b-1)$ -th query.

Each round of RANDOM-SPACE requires at most m queries, and may lead to two possible outcomes: One is a new domain value being discovered. The other is that RANDOM-SPACE reaches a valid or underflowing query *without* discovering any new domain value. Note that while this outcome never occurs for COUNT-DISCOVER, it might occur for RANDOM-SPACE because of the random construction of nested spaces. RANDOM-SPACE may be executed for multiple rounds to discover more domain values.

One can see that the randomization used by RANDOM-SPACE guarantees a positive probability for each domain value to be discovered. Nonetheless, the main problem of it occurs when the distribution of tuples in the database is severely skewed - e.g., when almost all tuples have the same values for A_1, A_2 , etc. In this case, a large number of queries may be wasted on a small number of tuples, rendering it unlikely for RANDOM-SPACE to unveil many domain values.

RANDOM-COUNT-SPACE: The main problem of RANDOM-SPACE can be solved by leveraging the COUNT information provided by the interface. In particular, RANDOM-COUNT-SPACE constructs the nested spaces in a similar fashion to RANDOM-SPACE, with the only exception that the value v in selection condition $A_i = v$ is no longer drawn uniformly at random from V_i . Instead, RANDOM-COUNT-SPACE first retrieves the COUNT of all $|V_i|$ possible queries that define the next search space (each corresponding to a different value of v). Then, it selects v in proportional to its corresponding COUNT. The construction of nested search spaces continues until either finding a new domain value or reaching a valid query. Note that since we now leverage COUNT information, an underflowing query will never be selected to construct the next search space.

An important property of RANDOM-COUNT-SPACE is that if we continue the search-space shrinking process until reaching a valid query, then the valid query returns each tuple in the database with roughly equal probability (precisely equal if $k = 1$, varies at most k times otherwise). This property explains why RANDOM-COUNT-SPACE solves the above-mentioned problem of RANDOM-SPACE. Nonetheless, RANDOM-COUNT-SPACE also has its own problem. In particular, if A_m is strongly correlated with A_1, A_2 , etc (e.g., $A_1, A_2 \rightarrow A_m$ forms a functional dependency), and all but a few tuples have the same value of A_m , then RANDOM-COUNT-SPACE may spend too many queries retrieving tuples fea-

turing the ‘‘popular’’ value of A_m (because the spaces it constructs most likely follow the ‘‘popular’’ value combination of A_1 and A_2), but fails to retrieve the other values. Interestingly, one can see that RANDOM-SPACE can handle this case pretty well, motivating our idea of mixing the two randomization approaches.

HYBRID-SPACE: We now consider a hybrid of RANDOM- and RANDOM-COUNT-SPACE to avoid the problems of both. In particular, HYBRID-SPACE starts with RANDOM-SPACE. It is repeated executed until none of the last $r_1 + 1$ times returns any new domain value, where r_1 is a small number (e.g., 0 to 5), the setting of which shall be discussed in the next subsection and in the experiments. At this time, we switch to RANDOM-COUNT-SPACE.

One round of RANDOM-COUNT-SPACE is performed. If it unveils a new domain value, then the *entire* process of HYBRID-SPACE is restarted by going back to the beginning of RANDOM-SPACE. If no new value is discovered, we repeat RANDOM-COUNT-SPACE for up to r_2 other rounds or until a new domain value is discovered, whichever happens first. Similar to r_1 , we shall discuss the setting of r_2 in the next subsection and in the experiments. If a new domain value is discovered, then the entire HYBRID-SPACE is restarted. Otherwise, If $r_2 + 1$ consecutive rounds of RANDOM-COUNT-SPACE fails to return any new domain value, we terminate HYBRID-SPACE. Algorithm 1 depicts the pseudocode for HYBRID-SPACE. Note that the input parameter in the pseudocode is designed for integration into RANDOM-COUNT-DISCOVER.

Algorithm 1 HYBRID-SPACE

```

1: Input parameter:  $q_0$ , set to SELECT * FROM D by default
2: repeat
3:   //Perform one round of RANDOM-SPACE
4:    $q \leftarrow q_0$ 
5:   while  $q$  overflows AND returns no new domain value do
6:     Select  $A_i$  not specified in  $q$ . Select  $v$  uniformly at random from the discovered domain of  $A_i$ 
7:      $q \leftarrow (q \text{ AND } A_i = v)$ . Issue  $q$  to learn new values.
8:   end while
9: until the last  $r_1 + 1$  rounds return no new domain value
10: repeat
11:   //Perform one round of RANDOM-COUNT-SPACE
12:    $q \leftarrow q_0$ 
13:   while  $q$  overflows AND returns no new domain value do
14:     Select  $A_i$  not specified in  $q$ . For each discovered  $v \in V_i$ , Query  $c(v) \leftarrow \text{COUNT of } (q \text{ AND } A_i = v)$ 
15:     Select  $v$  with probability proportional to  $c(v)$ 
16:      $q \leftarrow (q \text{ AND } A_i = v)$ . Issue  $q$  to learn new values.
17:   end while
18:   if  $q$  returns a new domain value then Goto 2.
19: until the last  $r_2 + 1$  rounds return no new domain value

```

4.2 HYBRID-SPACE Performance Guarantee

An important feature of HYBRID-SPACE is the performance guarantee it provides *independent* of the underlying data distribution. To illustrate such a guarantee, we consider an example where A_1, \dots, A_{m-1} each has a domain of size 10. Let there be $k = 50$, $n = 1,000,000$ tuples and $m = 10$ attributes in the database (note that n is readily available from the interface). Consider a parameter setting of $r_1 = r_2 = 0$. The following derivation shows how HYBRID-SPACE guarantees either a coverage of 46 values or a median recall of 500,000 tuples with only 4,036 queries.

Case 1: First consider the case where HYBRID-SPACE did not terminate before 4,036 queries were issued. Let w_B and w_T be the number of rounds RANDOM- and RANDOM-COUNT-SPACE

were performed, respectively. Since $r_1 = 0$, the number of discovered domain values is at least $w_B - 1$. Since A_1, \dots, A_{m-1} each has 10 domain values, an RANDOM-COUNT-SPACE shrinking process issues at most $9(m-1)$ queries. Thus, the $w_B + w_T$ finished rounds consume at least $4036 - (9(m-1) - 1) = 4046 - 9m$ queries because the final unfinished round could have issued at most $9(m-1) - 1$ queries. As such,

$$c \cdot w_B + 9(m-1) \cdot w_T \geq 4046 - 9m. \quad (7)$$

where c is the average number of queries issued by a round of RANDOM-SPACE during the execution. Since $r_2 = 0$, $w_B \geq w_T$. Thus, $(c + 9(m-1)) \cdot w_B \geq 4046 - 9m$. If HYBRID-SPACE did not discover at least 46 values, then $w_B \leq 46$ and therefore $c \geq (3956/46) - 9(m-1) = 5$.

A key step now is to prove that $c \geq 5$ indicates an absolute recall of at least 500,000. This is enabled by the following theorem which illustrates a positive correlation between the *absolute recall* γ and the *average query cost* c (per round) of RANDOM-SPACE.

THEOREM 4.1. *There is $\gamma \geq k \cdot 10^{c-1}$ with probability close to 1 when the number of rounds is sufficiently large.*

PROOF. Let there be a total of s rounds of RANDOM-SPACE performed. Consider the i -th round ($i \in [1, s]$). Let γ_i be the absolute recall of discovered domain values before the round begins. Given the discovered values, let Ω_i be the set of queries which may be the second-to-last query issued by this i -th round. One can see that each query in Ω_i must overflow and have all returned tuples featuring the already discovered domain values. Thus, we have $\gamma_i \geq k \cdot |\Omega_i|$, where $|\Omega_i|$ is the number of queries in Ω_i .

For each $q \in \Omega_i$, let $p(q)$ and $h(q)$ be the probability for q to be chosen in this round of RANDOM-SPACE and the number of predicates in v , respectively. Since each attribute in A_1, \dots, A_{m-1} has 10 possible values, we have $p(q) \leq 1/10^{h(q)}$. Thus,

$$E_{q \in \Omega_i}(k \cdot 10^{h(q)}) = \sum_{q \in \Omega_i} \frac{k \cdot 10^{h(q)}}{10^{h(q)}} = k \cdot |\Omega_i| \leq \gamma_i \quad (8)$$

where the expected value is taken over the randomness of nested space construction in RANDOM-SPACE.

Note that $k \cdot 2^x$ is a convex function of x . According to Jensen's inequality, $E_{q \in \Omega_i}(k \cdot 10^{h(q)}) \geq k \cdot 10^{E_{q \in \Omega_i}(h(q))}$. Also note that for any i , $\gamma_i \leq \gamma$. Thus, $k \cdot 10^{E_{q \in \Omega_i}(h(q))} \leq \gamma_i \leq \gamma$.

When s , the total number of rounds of RANDOM-SPACE, is sufficiently large, according to the central limit theorem, the probability of $E_{q \in \Omega_i}(h(q)) < c - 1$ for every $i \in [1, s]$ tends to 0. Thus, the probability of $\gamma \geq k \cdot 10^{c-1}$ tends to 1 when the number of drill-downs is sufficiently large. \square

According to the theorem, $c \geq 5$ indicates an absolute recall of at least $k \cdot 10^{5-1} = 500000$.

Case 2: Now consider the other case where HYBRID-SPACE terminates *before* 4,036 queries were issued. This indicates that the last round of RANDOM-COUNT-SPACE returns no new domain value. Let the absolute recall be $\alpha \cdot n$. With each round of RANDOM-COUNT-SPACE, the probability for the $(1 - \alpha) \cdot n$ "unrecalled" tuples to be retrieved is $1 - \alpha$. Thus, with Bayes' theorem, the posterior probability for $\alpha < 50\%$ given the empty discovery of the last round of RANDOM-COUNT-SPACE is

$$\Pr\{\alpha < 50\% | \text{empty discovery}\} < \frac{1/2 \cdot 1/2}{\int_0^1 p \, dp} = 1/2. \quad (9)$$

That is, the median absolute recall is at least 500000. In summary of both cases, HYBRID-SPACE either discovers at least 46 values, or has a median absolute recall of 500000.

Generic Performance Guarantee: More generally, we have the following algorithm:

THEOREM 4.2. *For a query budget of d over a top- k interface, HYBRID-SPACE achieves either an absolute coverage of m_0 values, or a median absolute recall of at least*

$$\min \left(k \cdot \prod_{i=1}^{\sigma} |V_i|, r_2 + 2 \sqrt{\frac{1}{2(r_2 + 2)} \cdot n} \right) \quad (10)$$

where

$$\sigma = \frac{d - \sum_{i=1}^{m-1} |V_i| + 1}{m_0 \cdot (r_1 + 1)} - \frac{r_2 + 1}{r_1 + 1} \cdot \sum_{i=1}^{m-1} |V_i|. \quad (11)$$

We do not include the proof due to the space limitation. One can see from the theorem that a suggest setting for r_1 and r_2 is $r_1 = r_2 = 0$, as it maximizes the lower bound derived in (10). We shall verify this suggested setting in the experiments.

Running Example: To discover the domain of Instrument from the NSF award search database based on pre-known domains of award amount, PI state, NSF organization, and field, HYBRID-SPACE requires at most 1563 queries to guarantee either the complete discovery of all domain values, or an absolute recall of $\min(4.12 \times 10^7, n/2)$.

4.3 RANDOM-COUNT-DISCOVER

We now integrate HYBRID-SPACE into COUNT-DISCOVER to produce RANDOM-COUNT-DISCOVER, our randomized (Monte Carlo) algorithm for discovering attribute domains. RANDOM-COUNT-DISCOVER starts with `SELECT * FROM D`, and uses the domain values discovered from the returned results to bootstrap HYBRID-SPACE. After that, RANDOM-COUNT-DISCOVER executes HYBRID-SPACE for multiple rounds, using the domain values discovered from previous rounds to bootstrap the next round.

In particular, at any time, RANDOM-COUNT-DISCOVER maintains a set of domain values that have been discovered but not used in the current round of HYBRID-SPACE. Let it be $V \subseteq \cup_{i=1}^n V_i$. At the start, V is empty. After every round of HYBRID-SPACE, the new domain values discovered are added to V . Then, in the next round of HYBRID-SPACE, we randomly select an attribute with domain values in V , say A_j and $\{v_{j1}, \dots, v_{jh}\} = V \cap V_j$, and start the search-space construction process with a selection condition of $A_j = v$ where $v \in \{v_{j1}, \dots, v_{jh}\}$. We then remove v from V . RANDOM-COUNT-DISCOVER terminates when V is empty or the query budget is exhausted.

Algorithm 2 RANDOM-COUNT-DISCOVER

- 1: $q_0 \leftarrow \text{SELECT * FROM D}, V \leftarrow \phi$.
 - 2: **repeat**
 - 3: Execute HYBRID-SPACE(q_0)
 - 4: $V \leftarrow$ domain values discovered by HYBRID-SPACE
 - 5: Arbitrarily select j and v such that $v \in V \cap V_j$
 - 6: $q_0 \leftarrow \text{SELECT * FROM D WHERE } A_j = v$
 - 7: $V \leftarrow V \setminus v$
 - 8: **until** $V = \phi$ or the query budget is exhausted
-

5. ATTRIBUTE DOMAIN DISCOVERY FOR ALERT INTERFACES

We now extend our results to ALERT interfaces which does not offer COUNT. Since the information provided by an ALERT interface is a proper subset of that provided by the corresponding

COUNT interface, the results in §3.2 already shows the impracticability of deterministic algorithms. Our results in this section further shows that ALERT interfaces require an even higher lower bound on query cost. To efficiently discover attribute domains, we develop RANDOM-ALERT-DISCOVER, a randomized algorithm which extends RANDOM-COUNT-DISCOVER by augmenting a query answer returned by an ALERT interface with an approximate COUNT estimated from historic query answers.

5.1 Achievability for Deterministic Algorithms

For ALERT interfaces, the achievability condition for k is exactly the same as the one for COUNT interfaces - which is shown in Theorem 3.4 - as one can easily verify that the proof of Theorem 3.4 does not use any COUNT returned by the interface. For the condition on query cost, however, the lower bound is significantly higher for ALERT interfaces, as shown by the following theorem.

THEOREM 5.1. *For given k and m , there exists a database and a scoring function such that no deterministic algorithm can complete the discovery of all domain values that can be discovered from the top- k ALERT interface without incurring a worst-case query cost of $\prod_{i=h+1}^m |V_{d_i}|$, where d_1, \dots, d_m is the permutation of $1, \dots, m$ which satisfies $|V_{d_1}| \geq \dots \geq |V_{d_m}|$, and h is the minimum value which satisfies $k \leq 1 + \prod_{i=1}^h (|V_{d_i}| - 1)$.*

PROOF. We prove by induction. In particular, we start with constructing a database instance D_1 , and then show that for any $b < \prod_{i=h+1}^m |V_{d_i}|$, if a deterministic algorithm uses exactly b queries q_1, \dots, q_b to discover all attribute domains from D_1 , then there must exist another database instance D_2 which (1) provides the exact same query answers for q_1, \dots, q_{b-1} , such that the deterministic algorithm will always issue q_b next, and (2) ensures that q_1, \dots, q_b cannot discover all attribute domains. One can see that if this is proved, then any deterministic algorithm which discovers all attribute domains must have a query cost of at least $\prod_{i=h+1}^m |V_{d_i}|$.

We construct D_1 with $1 + k \cdot \prod_{i=h+1}^m |V_{d_i}|$ tuples t_1, \dots, t_{2^m-1} in the following manner. First, for each A_{d_i} ($i \in [1, h]$), we arbitrarily choose a domain value $v_i \in A_{d_i}$. Then, for each value combination of $V_{d_{h+1}}, \dots, V_{d_m}$, we include k other tuples with $A_{d_{h+1}}, \dots, A_{d_m}$ defined by the value combination and $A_{d_i} \neq v_{d_i}$ for all $i \in [1, h]$. It is always possible to find such k tuples because of our assumption that $k \leq 1 + \prod_{i=1}^h (|V_{d_i}| - 1)$. The one additional tuple in D_1 is t which satisfies $A_{d_i} = v_{d_i}$ for all $i \in [1, h]$ and has an arbitrary value combination for $A_{d_{h+1}}, \dots, A_{d_m}$. The scoring function is designed such that t has the lowest score.

Note that the deterministic algorithm must unveil t with the first b queries. Without loss of generality, we assume that the b -th query returns t . An important observation here is that t can only be unveiled by a query which has no predicate for A_{d_1}, \dots, A_{d_h} (because v_1, \dots, v_h could not have been discovered before t is unveiled) and one predicate for each of $A_{d_{h+1}}, \dots, A_{d_m}$ (because of k). If $b < \prod_{i=h+1}^m |V_{d_i}|$, there must exist at least one value combination of $A_{d_{h+1}}, \dots, A_{d_m}$ which has not yet been (fully) specified by any issued query. We then construct D_2 by changing the values of $A_{d_{h+1}}, \dots, A_{d_m}$ of t to this not-yet-specified combination. One can see that the answers of q_1, \dots, q_{b-1} over D_2 is exactly the same as D_1 . Nonetheless, q_b no longer returns t , mandating a query cost of at least $b + 1$ for attribute domain discovery over D_2 . \square

Running Example: With the NSF award search example, the lower bound on worst-case query cost over a top-50 ALERT interface is 9.40×10^{15} , orders of magnitude larger than even the upper bound for COUNT interface.

THEOREM 5.2. *For given k and m -attribute, there exists an algorithm which requires at most $h - 1 + \prod_{i=1}^{h-1} |V_{d_i}|$ queries to discover the domains of at least h (out of the m) attributes, where d_1, \dots, d_m is the permutation of $1, \dots, m$ which satisfies $|V_{d_1}| \geq \dots \geq |V_{d_m}|$.*

The following description of ALERT-DISCOVER serves as the proof for this theorem.

5.1.1 ALERT-DISCOVER

The deterministic COUNT-DISCOVER is capable of shrinking the search space significantly at each step because it can infer the COUNT of a domain value yet to be discovered from COUNTs returned by other queries. ALERT interfaces do not provide such luxury. In particular, for the discovery of V_m , there is no direct way to check whether an unknown domain value in V_m occurs in tuples satisfying an overflowing query such as q_3 : `SELECT * FROM D WHERE $A_1 = 0_1$` , unless q_3 returns the domain value in its top- k result.

Despite of the lack of COUNT information, we can still shrink the search space with an ALERT interface, though the shrinking may not be as significant as in COUNT-DISCOVER. For example, if the above-mentioned q_3 turns out to be valid or underflowing without returning any unknown domain value in V_m , we can safely remove from the search space $S_1(V_m)$ all values which satisfy q_3 - a reduction of size $\prod_{i=2}^{m-1} |V_i|$ - while still ensuring that the reduced search space contains at least one tuple in the database which has an unknown value of V_m .

ALERT-DISCOVER, our deterministic algorithm for ALERT interfaces, exactly follows this strategy to construct a series of nested spaces $S_1(V_m), S_2(V_m), \dots$, for the discovery of V_m . To decide which query to issue next, ALERT-DISCOVER uses the following simple rule: Suppose that the complete domains of h attributes have been discovered, say A_1, \dots, A_h without loss of generality (note that $h = 1$ at the start of the algorithm). The next query ALERT-DISCOVER issues is an h -predicate query with one conjunctive condition for each attribute A_i ($i \in [1, h]$). The value v_i specified for A_i in the query is determined as follows: First, the value combination $\langle v_1, \dots, v_h \rangle$ must lead to a query that cannot be answered based solely upon the historic query answers - i.e., no previously issued query is formed by a subset of $A_1 = v_1, \dots, A_h = v_h$ and returns valid or underflow. Within the value combinations which satisfy this condition, we select the highest-ordered one according to an arbitrary global order. The above process is repeated until all domains are discovered.

To understand how such a query-issuing plan leads to the shrinking of nested search spaces, consider the three possible outcomes of an above-constructed query q - underflow, valid, or overflow. If q underflows or is valid, then we can construct the next search space by removing from the current one all value combinations that satisfy q - a reduction of size $\prod_{i=h+1}^{m-1} |V_i|$ where h is the number of predicates in q . On the other hand, if q overflows, then we can always discover the complete domain of at least one additional attribute because $k \geq 1 + \prod_{i=1}^h (|V_i| - 1)$. One can see that if ALERT-DISCOVER has issued at least m queries with the second outcome, then it must have discovered the domains for all m attributes. The first outcome, on the other hand, guarantees the reduction of search space. Theorem 5.2 follows accordingly.

5.2 RANDOM-ALERT-DISCOVER

A simple approach to enable aggregate estimation is to directly integrate RANDOM-COUNT-DISCOVER with HD-UNBIASED-AGG, the existing aggregate estimator for hidden databases [8]. Intuitively, since RANDOM-COUNT-DISCOVER starts with RANDOM-

SPACE, one can use queries answers received by RANDOM-SPACE to estimate each COUNT value required by the subsequent execution of RANDOM-COUNT-SPACE. The estimations will become more accurate over time after more queries are issued by RANDOM-SPACE or RANDOM-COUNT-SPACE).

Nonetheless, such a direct integration has a key problem: The existing HD-UNBIASED-AGG can only generate a COUNT estimation from a *valid* query answer. Most queries issued by RANDOM-COUNT-SPACE, however, are likely to overflow as the shrinking of search space only continues (i.e., with “narrower”, and possibly valid, queries being issued) if no new domain value can be discovered from the previously issued overflowing queries - an unlikely event especially at the beginning of RANDOM-COUNT-DISCOVER. As a result, the integration with HD-UNBIASED-AGG faces a dilemma: either uses only a few valid queries and suffers a high estimation variance, or issues a large number of extra queries to “narrow down” the queries issued by RANDOM-COUNT-SPACE (by adding conjunctive conditions to the queries) to valid ones. Either way, the large number of overflowing queries issued by RANDOM-COUNT-SPACE would be wasted.

To address this problem, we propose a step-by-step estimation process which can generate COUNT estimations based the the results of all historic queries, including the overflowing ones. For the ease of understanding, we consider an example of estimating `SELECT COUNT(*) FROM D WHERE $A_1 = v_1$` , denoted by `COUNT(v_1)`. Handling of other COUNT queries immediately follows. Consider one round of RANDOM-SPACE or RANDOM-COUNT-SPACE during which b queries q_1, \dots, q_b are issued in order. Recall that q_1 is `SELECT * FROM D`. q_b may be overflowing, valid or underflowing. Without loss of generality, let A_1, \dots, A_{i-1} be the attributes involved in the selection conditions of q_i ($i \in [2, b]$). Let p_i be the *transition probability* from q_i to q_{i+1} - i.e., the probability for q_{i+1} to be selected (out of all possible values of A_i) after q_i is issued. For example, $p_i = 1/|V_i|$ for RANDOM-SPACE, and may differ (based on the currently estimated COUNTs) for RANDOM-COUNT-SPACE. We estimate `COUNT(v_1)` from all b queries with the following two steps:

- We first choose i.i.d. uniformly at random $v_i \in V_i$ for each $i \in [b, m - 1]$, and then construct and issue a (conjunctive) query q_F formed by appending conditions ($A_b = v_b$) AND \dots AND ($A_{m-1} = v_{m-1}$) to q_c .
- We then return the following estimation of `COUNT(v_1)`:

$$\omega = \left(\sum_{i=1}^b \frac{S_{v_1}(\beta_i)}{\prod_{j=1}^{i-1} p_j} \right) + \frac{S_{v_1}(\beta_F) \cdot \prod_{j=b}^m |V_j|}{\prod_{j=1}^{b-1} p_j} \quad (12)$$

where $\beta_i = \{t | t \in q_i, t \notin (q_1 \cup \dots \cup q_{i-1})\}$, $\beta_F = \{t | t \in q_F, t \notin (q_1 \cup \dots \cup q_c)\}$, and $S_{v_1}(\cdot)$ stands for the result of applying `COUNT(v_1)` over a set of tuples. Note that here q_i stands for the set of tuples returned by query q_i . We assume $\prod_{j=1}^{i-1} p_j = 1$ when $i = 1$, and $S_{v_1}(\phi) = 0$.

One can see that each query in q_1, \dots, q_b contributes an additive component to the final estimation ω - e.g., q_i contributes $S_{v_1}(\beta_i) / \prod_{j=1}^{i-1} p_j$, which is essentially an estimated COUNT of tuples that satisfy $A_1 = v_1$, can be returned by a query with predicates on A_1, \dots, A_i , but cannot be returned by a query with predicates only on A_1, \dots, A_{i-1} . This is why we call the idea “step-by-step estimation”. In terms of query cost, step-by-step estimation reuses all queries issued by RANDOM-SPACE or RANDOM-COUNT-SPACE. In addition, it issues at most one additional query q_F per round (of search-space shrinking). It is easy to verify that the proof of unbiasedness for HD-UNBIASED-AGG [8] remains valid with step-by-step estimation.

6. EXPERIMENTAL RESULTS

6.1 Experimental Setup

1) *Hardware and Platform*: All our experiments were performed on a 2.6GHz Intel Core 2 Duo machine with 2GB RAM and Windows XP OS. All algorithms were implemented in C++.

2) *Dataset*: We conducted our experiments on two real-world datasets. One is the NSF award search database which has been used as a running example throughout the paper. We tested our algorithms over the real-world web interface of NSF award search by issuing queries through the “search all fields” option of NSF award search, available at <http://www.nsf.gov/awardsearch/tab.do?dispatch=4>. Since this interface uses HTTP GET to specify query selection conditions, we issue search queries through the interface by appending search conditions to the URL. Since NSF award search returns up to 50 awards on a returned page, we set $k = 50$, but also tested cases where k varies from 10 to 40 by truncating the returned results. As mentioned in §2.1, NSF award search is an ALERT interface. We conducted experiments with all 9 attributes listed in §2.1, with objective being the discovery of domain values for these attributes.

While experiments over the NSF award search database demonstrates the practical impact of our approach, to more thoroughly evaluate its effectiveness we need access to certain *ground truth*, e.g., the total number of tuples in the database or the total number of domain values to be discovered, which we do not have over NSF award search. Thus, we also conducted experiments over a local hidden database built from a Texas inpatient discharge dataset published by the Texas Department of Health [22]. There are 707,735 tuples and 260 attributes. We used all tuples but randomly chose 23 attributes with domain size ranging from 2 to 19028, in order to better simulate the number of attributes commonly available from a hidden database. We constructed the top- k interface with $k = 100$ and a random scoring function, but also tested cases where k varies from 100 to 400. We considered both COUNT and ALERT interfaces for this database.

3) *Algorithms*: We tested seven algorithms: the crawling-based DFS and BFS (as baseline), our main algorithms RANDOM-COUNT-SPACE and RANDOM-ALERT-SPACE, a main component of them HYBRID-SPACE, as well as stand-alone RANDOM-SPACE and RANDOM-COUNT-SPACE. The only parameters required are r_1 and r_2 for HYBRID-SPACE, which is in turn used by RANDOM-COUNT-SPACE and RANDOM-ALERT-SPACE. We set $r_1 = r_2 = 0$ by default but also tested other values which justified our default settings. To test HYBRID-SPACE and the standalone RANDOM- and RANDOM-COUNT-SPACE, we considered the case where all attribute domains except the largest one have been completely discovered, and set the goal to be the discovery of the largest domain with 19028 values.

3) *Performance Measures*: For query cost, we focused on the number of queries issued through the web interface of the hidden database. For estimation accuracy, we tested both measures discussed in §2.2: absolute recall and coverage. For the local hidden database, we also tested the relative recall, i.e., absolute recall divided by the total number of tuples in the database, and relative coverage, i.e., absolute coverage for an attribute domain divided by its domain size (ground truths that we only have access to over the local database), to provide an intuitive demonstrate of the effectiveness of our algorithms.

6.2 Experimental Results

We compared the performance of RANDOM-ALERT-DISCOVER, DFS and BFS over the NSF award search database when no attribute domain is pre-known. Figures 1 and 2 depict the change of recall and coverage with query cost, respectively. Note that in

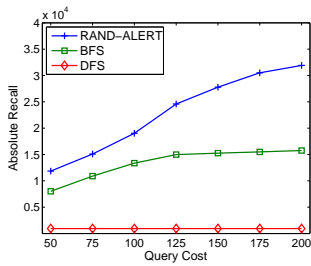


Figure 1: Recall vs query cost

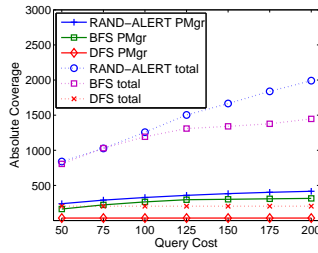


Figure 2: Coverage vs query cost

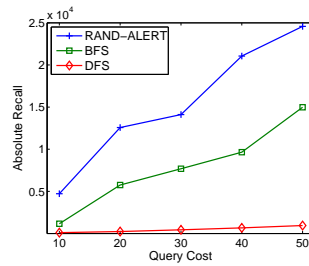


Figure 3: Recall vs k

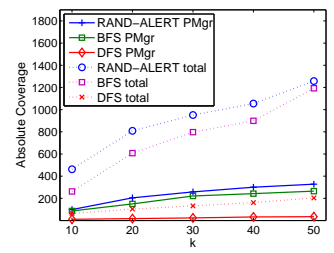


Figure 4: Coverage vs k

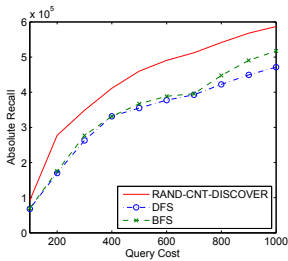


Figure 5: Recall vs query cost

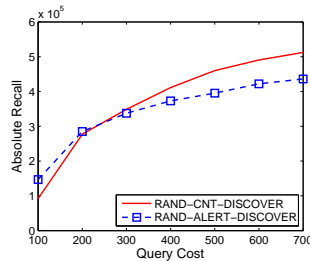


Figure 6: ALERT vs COUNT

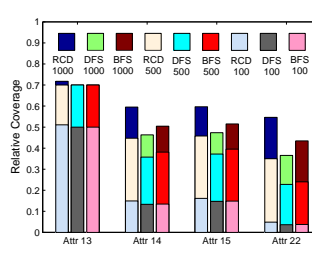


Figure 7: Relative coverage

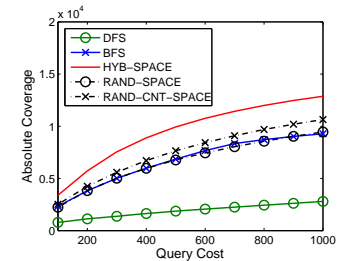


Figure 8: HYB-SPACE coverage

Figure 2, we present both the coverage for attribute Program Manager and the SUM of coverage for all attributes. One can see from the figures that RANDOM-ALERT-DISCOVER provides a significantly higher recall and coverage than both baseline algorithms.

We also tested the NSF award search database with varying k , the only parameter we can control over its interface. Figures 3 and 4 depict the change of recall with k when 100 queries were issued. One can see that as suggested by intuition, the larger k is, the more recall all three algorithms will achieve. In addition, RANDOM-ALERT-DISCOVER significantly outperforms DFS and BFS for all settings of k .

To provide a more thorough evaluation of the algorithms, we turned our attention to the local hidden database. We compared the tradeoff between recall and query cost for RANDOM-COUNT-DISCOVER, DFS and BFS. The results are shown in Figure 5. We also compared the performance of RANDOM-COUNT-DISCOVER with RANDOM-ALERT-DISCOVER, with results shown in Figure 6. One can see that RANDOM-COUNT-DISCOVER significantly outperforms DFS and BFS, while the difference between recall of RANDOM-COUNT-DISCOVER and RANDOM-ALERT-DISCOVER is not significant.

An interesting (and somewhat counter-intuitive) observation here is that when the query cost is small (e.g., 100), RANDOM-ALERT-DISCOVER *even outperforms* RANDOM-COUNT-DISCOVER. This is because RANDOM-ALERT-DISCOVER leverages previously issued queries estimate the COUNT of each discovered domain value (to decide which to use in the shrinking of search space), instead of actually querying the COUNTs as in RANDOM-COUNT-DISCOVER. This leads to a significant saving of query cost in the beginning. But when more queries are issued, many of the saved queries may be (randomly) selected for shrinking search space and issued anyway. As such, the advantage provided by precise COUNTs becomes more evident and helps increase the recall.

To further demonstrate the coverage achieved by RANDOM-COUNT-DISCOVER, we evaluated the relative coverage for all attributes and present in Figure 7 the four attributes with the *minimum*

(i.e., worst) relative coverage when 100, 500 and 1000 queries have been issued. One can see that even for these worst-case attributes, RANDOM-COUNT-DISCOVER can still unveil more than half of the domain values for every attribute with a query cost of only 1000. In addition, RANDOM-COUNT-DISCOVER outperforms both DFS and BFS on all four attributes.

Since HYBRID-SPACE is the key component of both RANDOM-COUNT-DISCOVER and RANDOM-ALERT-DISCOVER, we compared experiments on HYBRID-SPACE alone with the aforementioned experimental setup to justify our design of mixing RANDOM-SPACE with RANDOM-COUNT-SPACE. In particular, we compared the performance of HYBRID-SPACE, DFS, BFS, as well as RANDOM-SPACE and RANDOM-COUNT-SPACE. Figures 8 and 9 depict the absolute coverage and recall achieved by all 5 algorithms, respectively. One can see that HYBRID-SPACE significantly outperforms all other approaches on both measures.

We also tested the five algorithms with varying database size n . Figures 10 and 11 depict the relative recall and absolute coverage for a query cost of 100, respectively. Note that while the relative recall naturally decreases for a larger database, the absolute coverage actually increases. Also, our approach consistently outperforms the others over all database sizes, again justifying the mixing of RANDOM- and RANDOM-COUNT-SPACE.

We also studied how the parameters r_1 and r_2 of HYBRID-SPACE affects its performance. Figure 12 shows the change of absolute recall when r_1 and r_2 varies from 0 to 4 (absolute coverage shows a similar trend). One can see that while r_2 does not appear to have a significant impact on the completeness of attribute domain discovery, r_1 should be set as small as possible - justifying our default setting of $r_1 = 0$.

7. RELATED WORK

Information Integration and Extraction for Hidden databases:

A significant body of research has been done on information integration and extraction over deep web data sources such as hidden databases - see tutorials in [6, 11]. Nonetheless, to the best of

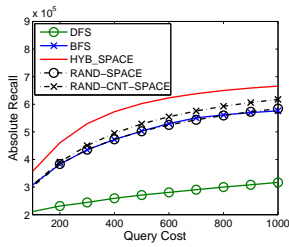


Figure 9: HYB-SPACE recall

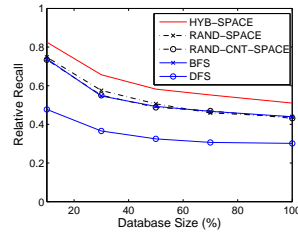


Figure 10: Recall vs DB size

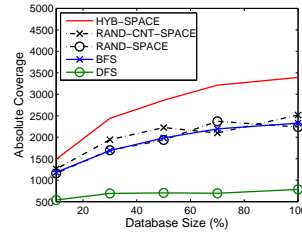


Figure 11: Coverage vs DB size

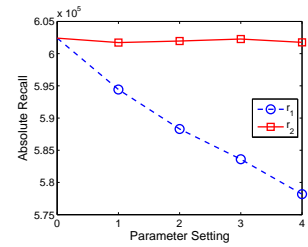


Figure 12: Varying r_1, r_2

our knowledge, the only prior work which directly tackles the attribute domain discovery problem is [20]. In particular, it proposes a crawling-based technique, the disadvantage of which has been extensively discussed in §2.3. Much other work though is related but orthogonal to attribute domain discovery. Since there is no space to enumerate all related papers, we only list a few examples closely related to this paper. Parsing and understanding web query interfaces has been extensively studied (e.g., [12, 23]). The mapping of attributes across different web interfaces has also been addressed (e.g., [15]). Also related is the work on integrating query interfaces for multiple web databases in the same topic-area (e.g., [13, 14]). Our paper provides results orthogonal to these existing techniques as it represents the first formal study on attribute domain discovery over hidden databases.

Data Analytics over Hidden Databases: There has been prior work on crawling, sampling, and aggregate estimation over the hidden web, specifically over text [2, 3] and structured [20] hidden databases and search engines [1, 19, 21]. In particular, sampling-based methods were used for generating content summaries [5, 16, 17], processing top- k queries [4], etc. Prior work (see [8] and references therein) considered sampling and aggregate estimation over structured hidden databases. A key difference between these techniques and this paper is that the prior techniques assume full knowledge of all attribute domains, while this paper aims to integrate domain discovery with aggregate estimation. As we demonstrated in §6, our integrated approach significantly outperforms the direct application of previous techniques [8] after domain discovery.

8. CONCLUSION

In this paper we have initiated a formal study of the discovery of attribute domains through the web interface of a hidden database. We investigated the achievability of deterministic algorithms for both COUNT and ALERT interfaces, and proposed query-efficient randomized algorithms for attribute domain discovery based on the idea of constructing nested search spaces. We provided theoretical analysis and extensive experimental studies over real-world datasets to illustrate the effectiveness of our approach.

9. REFERENCES

- [1] Z. Bar-Yossef and M. Gurevich. Efficient search engine measurements. In *WWW*, 2007.
- [2] Z. Bar-Yossef and M. Gurevich. Mining search engine query logs via suggestion sampling. In *VLDB*, 2008.
- [3] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. In *WWW*, 1998.
- [4] N. Bruno, L. Gravano, and A. Marian. Evaluating top- k queries over web-accessible databases. In *ICDE*, 2002.
- [5] J. Callan and M. Connell. Query-based sampling of text databases. *ACM TOIS*, 19(2):97–130, 2001.
- [6] K. Chang and J. Cho. Accessing the web: From search to integration. In *Tutorial, SIGMOD*, 2006.
- [7] A. Dasgupta, G. Das, and H. Mannila. A random walk approach to sampling hidden databases. In *SIGMOD*, 2007.
- [8] A. Dasgupta, X. Jin, B. Jewell, N. Zhang, and G. Das. Unbiased estimation of size and other aggregates over hidden web databases. In *SIGMOD*, 2010.
- [9] A. Dasgupta, N. Zhang, and G. Das. Leveraging count information in sampling hidden databases. In *ICDE*, 2009.
- [10] A. Dasgupta, N. Zhang, and G. Das. Turbo-charging hidden database samplers with overflowing queries and skew reduction. In *EDBT*, 2010.
- [11] A. Doan, R. Ramakrishnan, and S. Vaithyanathan. Managing information extraction. In *Tutorial, SIGMOD*, 2006.
- [12] E. Dragut, T. Kabisch, C. Yu, and U. Leser. A hierarchical approach to model web query interfaces for web source integration. In *VLDB*, 2009.
- [13] E. Dragut, C. Yu, and W. Meng. Meaningful labeling of integrated query interfaces. In *VLDB*, 2006.
- [14] B. He and K. Chang. Statistical schema matching across web query interfaces. In *SIGMOD*, 2003.
- [15] B. He, K. Chang, and J. Han. Discovering complex matchings across web query interfaces: A correlation mining approach. In *KDD*, 2004.
- [16] 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.
- [17] P. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *VLDB*, 2002.
- [18] R. Khare, Y. An, and I.-Y. Song. Understanding deep web search interfaces: A survey. *SIGMOD Record*, 39(1), 2010.
- [19] K. Liu, C. Yu, and W. Meng. Discovering the representative of a search engine. In *CIKM*, 2002.
- [20] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB*, 2001.
- [21] M. Shokouhi, J. Zobel, F. Scholer, and S. Tahaghoghi. Capturing collection size for distributed non-cooperative retrieval. In *SIGIR*, 2006.
- [22] Texas Department of State Health Services. User manual of Texas hospital inpatient discharge public use data file, 2008. <http://www.dshs.state.tx.us/thcic/Hospitals/HospitalData.shtm>.
- [23] Z. Zhang, B. He, and K. Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In *SIGMOD*, 2004.