Aggregate Suppression for Enterprise Search Engines

Mingyang Zhang
George Washington University
Washington, DC 20052, USA
mingyang@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 enterprise websites provide search engines to facilitate customer access to their underlying documents or data. With the web interface of such a search engine, a customer can specify one or a few keywords that he/she is interested in; and the search engine returns a list of documents/tuples matching the user-specified keywords, sorted by an often-proprietary scoring function.

It was traditionally believed that, because of its highly-restrictive interface (i.e., keyword search only, no SQL-style queries), such a search engine serves its purpose of answering individual keyword-search queries without disclosing big-picture aggregates over the data which, as we shall show in the paper, may incur significant privacy concerns to the enterprise. Nonetheless, recent work on sampling and aggregate estimation over a search engine's corpus through its keyword-search interface transcends this traditional belief. In this paper, we consider a novel problem of suppressing sensitive aggregates for enterprise search engines while maintaining the quality of answers provided to individual keyword-search queries. We demonstrate the effectiveness and efficiency of our novel techniques through theoretical analysis and extensive experimental studies.

Categories and Subject Descriptors

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

General Terms

Algorithms, Measurement, Performance

Keywords

Search Engine, Sampling, Aggregate Suppression

*Partially supported by NSF grants 0852674, 0915834, 1117297 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.

SIGMOD '12, May 20–24, 2012, Scottsdale, Arizona, USA.

1. INTRODUCTION

Enterprise Search Engine: With web-scale search engines (e.g., Google, Bing) becoming the de facto way for web users to locate and access resources of interest, many enterprises have coped with the trend by offering their customers *enterprise search engines* for accessing the large amounts of documents or (structured) data inside the enterprise's website. Examples of such enterprise search engines range from keyword-based product search provided by online retailers such as Amazon.com, to case search provided by many government agencies such as the US patent office, and to article search provided by almost all online content providers (e.g., Washington Post).

In general, the web interface of such a search engine offers a text-box input through which a customer can specify one or a few keywords that he/she is interested in. The search engine will return a list of documents or tuples that match the user-specified keywords[1]. Because of the length limit of a return web page, not all matching documents/tuples may be returned. Instead, the top-k results are selected according to a scoring function often proprietary to the enterprise, and then returned to the customer.

Privacy Concerns on Sensitive Aggregates: While an enterprise search engine is designed to answer individual search queries specified by its customers, recent sampling-based techniques have been developed ([9,26]) that allow third party applications to issue queries to the search interface (queries are randomly selected from a query pool), and from the answers piece together big-picture aggregate information about the underlying corpus. Although in some cases it is advantageous to have such aggregate information disclosed, there are others for which an enterprise would not willingly like to disclose aggregate information through its search engine. To understand why, consider the following examples.

- *Commercial Competition:* Many online retailers allow a customer to search for keywords in product reviews left by other customers, so that the customer can quickly locate products with desired properties. Nonetheless, the retailer may not want a competitor to learn certain aggregate information which places the retailer in a disadvantageous position for competition - e.g., if the competitor learns that the total number of products in the retailer's website which have "poor quality" in a user review is twice as large as the number in the competitor's own website, then the competitor may use the in-

[1]For keyword search over structured data, note that while there has been recent work [16] on supporting keyword search over the JOIN of multiple tables, most real-world search engines simply consider each tuple as a document consisting of all attribute values of the tuple, and process the keyword-search query in (almost) the same way as search over unstructured documents.

formation against the retailer through boasting "fewer poor-quality products", etc., in an advertisement campaign.

- *Government Concern:* Many government agencies support individual searches for legal compliance purposes, but may not be willing to disclosure aggregate information that can lead to speculative rumors or negative publicity. For example, US Patent and Trademark Office supports keyword search for patents, with the patent examiner's name displayed on the description of each returned patent. Nonetheless, the office may not want to disclose aggregates such as the total number of patents approved by a particular examiner in 2010, because such information may allow a third party to infer (based on the common workload of a patent examiner) sensitive information such as the approval rate of the examiner - which may lead to negative publicity for the office.

To address the search engine owners' concerns on sensitive aggregates, we define and study a novel problem of *search-engine aggregate suppression* in this paper - i.e., our objective is to suppress a third party's access to sensitive aggregates over a search engine's corpus without affecting the utility of the search engine to normal search users. This problem stands in sharp contrast to existing studies on privacy protection for search engines which mostly focus on protecting the privacy of individual users' search query logs [4, 18] or preventing the search engine from learning a user's real search query [15, 23].

Note that while the problem of aggregate suppression (and, in fact, our proposed techniques) may also apply to web-scale search engines such as Google and Bing, in this paper we focus our attention on enterprise search engines. This is mainly because the privacy concerns of aggregate disclosure is more acute in the case of the enterprise owners (as our earlier motivating examples show), whereas the corpus of a web search engine is essentially a collection of publicly crawlable web pages and aggregate disclosures may not be as significant for web search engine owners.

Technical Challenge: A seemingly promising approach to achieve aggregate suppression over a search engine is to directly apply the two existing techniques for suppressing aggregates over structured databases - (1) the insertion of dummy tuples [12] and (2) the (randomized) generalization of tuple attribute values [17]. Nonetheless, neither technique applies to unstructured documents in a search engine's corpus. In particular, generating a dummy document which is not (at least not easily) recognizable as dummy by an adversary is significantly more difficult than generating a structured dummy tuple (which "looks" real). Likewise, there is no concept hierarchy defined over an unstructured document - an important ingredient of the generalization technique [17] - so the generalization technique does not apply either. Thus, the problem of search-engine aggregate suppression calls for the development of novel privacy preservation techniques.

Solution Space for Aggregate Suppression: One can partition the solution space for aggregate suppression into two main types of techniques: One is to revise the keyword search interface of the search engine - e.g., by disallowing certain queries and/or returning snippets instead of entire documents unless a user signs in with a unique identity. While there are fee-based search engines in practice which enforce such limitations to attract paying customers (e.g., reverse phone lookup at *http://www.whitepages.com/*), we do not consider this approach for our purpose of aggregate suppression because of the difficulty on evaluating the loss of service quality provided to normal search users - which by itself might be an interesting problem for the Human-Computer Interaction (HCI) research community.

The other type of solution, which we advocate in this paper, is to revise the keyword-query processing mechanism by changing (a small number of) documents returned for a given query. To this end, we may add dummy documents, revise the content of returned documents, or hide certain documents from the query answer. As discussed above, we do not consider adding dummy documents due to the difficulty of creating dummy contents that look "reasonable" to an adversary. For the same reason, we do not consider revising the content of a returned document. Thus, we focus on the last option - i.e., hiding (a small number of) returned documents from certain query answers.

Outline of Technical Results: To understand our main idea, we first consider two simple techniques which achieve either aggregate suppression or search utility but not both at the same time. Consider as an example two search engines A_1 and A_2 with corpus sizes n and $2n$, respectively. Suppose that A_1 is a simple random sample (without replacement) from A_2. Consider an objective of suppressing access to the total size of each corpus (i.e., suppressing access to the aggregate COUNT(*)), such that an adversary cannot tell which corpus is larger. A brute-force approach of doing so through document hiding is to first remove n documents from the corpus of A_2, and then answer each keyword query over A_2 from the shrunken corpus. This technique perfectly protects privacy by making the two COUNTs indistinguishable, but may lead to poor search utility over A_2 because half of the documents in (the original) A_2 will never be returned by any query.

Another seemingly simple idea is to perform document hiding at run-time. In particular, when a keyword query q is received by A_2, one first identifies the number of documents in A_2 which match q, denoted by $|q|$. Then, one selects and removes half of these matching documents. The result - i.e., $|q|/2$ documents - are then returned as the query answer (subject to top-k selection according to the scoring function). This technique addresses the utility problem of the brute-force technique because, even if a document is severed from one query answer, it might still appear in another one. This allows the number of documents recallable by keyword queries to approach $2n$. It might appear that this approach suppresses COUNT(*) as well, because the expected number of documents returned by a given query is the same over the two corpora once run-time document hiding is applied over A_2.

However, this approach actually fails on aggregate suppression. To understand why, consider a pool of queries formed by all English words and a document X that appears in both A_1 and A_2. For the ease of understanding, suppose that k is sufficiently large such that no query in the pool is subject to the top-k restriction on its returned results. Before run-time document hiding, the number of queries in the pool which return X is the same over A_1 and A_2. Nonetheless, once run-time document hiding is applied (over A_2), the number of queries which return X over A_2 is (expectedly) reduced by half - making it easy for an adversary distinguish between A_1 and A_2 by first finding all words in X and then issuing them over A_1 and A_2, respectively, to find out which ones return X.

To address the problems of these two simple techniques, we develop the aggregate suppression technique, AS-SIMPLE, which suppresses aggregates by carefully adjusting both *query degree*, i.e., the number of documents matched by a query, and *document degree*, i.e., the number of queries matching a document, through document hiding at run-time (i.e., when a query is received). In terms of aggregate suppression, we prove that AS-SIMPLE can thwart a large class of sampling-based aggregate estimation algorithms including all the existing ones [9, 26]. In terms of search

engine utility, we find (somewhat surprisingly) that most keyword queries issued by real-world users[2] only see minimum changes to their answers. Intuitively, this is because most of these real-world queries are *overflowing* ones which have their answers truncated by the top-k restriction. As such, we can perform the necessary adjustments to query/document degrees while hardly affecting the k documents that are actually returned to the user.

Although AS-SIMPLE can thwart all existing aggregate-estimation attacks, during the course of our investigations we found a new (non-sampling-based) attack against AS-SIMPLE, which works by issuing *highly correlated* queries - i.e., those which return significantly overlapping, if not the same, sets of documents. To thwart such a correlation-based attack, we develop AS-ARBI[3] which, upon receiving a query, first calls AS-SIMPLE as a subroutine to generate an initial query answer, and then post-processes it by appending historic query answers which match the new query. We show that AS-ARBI not only addresses the threat from correlation based attacks in an efficient manner, but also provides better search engine utility than AS-SIMPLE.

Summary of Contributions: Our contributions in the paper can be summarized as follows.

- We define the novel problem of aggregate suppression over a search engine's corpus.
- We develop two novel techniques, AS-SIMPLE and AS-ARBI, which are capable of suppressing access to sensitive aggregates while maintaining a high level of utility for individual search queries.
- Our contributions also include a comprehensive set of experiments on a real-world document corpus. The experiments validate the effectiveness of our techniques on (1) aggregate suppression against multiple existing aggregate estimation techniques, and (2) maintaining search engine utility for a real-world keyword query log.

The rest of this paper is organized as follows. We discuss the preliminaries in Section 2 and define our problem in Section 3. In Sections 4 and 5, we develop AS-SIMPLE and AS-ARBI, respectively, and present theoretical analysis on their ability of suppressing aggregates and maintaining search engine utility. We describe the experimental results in Section 6, followed by a brief review of related work in Section 7 and conclusions in Section 8.

2. PRELIMINARIES

In this section, we first introduce our model of an enterprise search engine, and then briefly review the existing attacks for aggregate estimation over a search engine's corpus.

2.1 System Model

Search Engine: Consider a search engine accessible by web users through a keyword-search interface. Let Θ be the search engine's corpus - i.e., the set of documents searchable through the interface. A user can search for documents in Θ by specifying a *search query* consisting of one or a few words. For the purpose of this paper, we consider a simple model of a search engine as follows: The interface is restricted to return up to k documents, where k is a predetermined small constant (e.g., 50 or 100). Thus, for a given query q, all documents matching q, i.e., $Sel(q)$, can be entirely returned iff there are at most k matching documents, i.e., $|Sel(q)| \leq k$. If the query *overflows* (i.e., $|Sel(q)| > k$), only the top-k documents in $Sel(q)$ are selected according to a scoring function (unknown to external users) and returned as the query answer. The interface also notifies the user that there is an overflow. At the other extreme, if the query is too specific and matches no document, we say that an *underflow* occurs. If there is neither overflow nor underflow, we have a *valid* query result. We consider deterministic query processing - i.e., a query executed again will produce the same results.

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

Another restriction commonly enforced by search engines is a limit on the number of (keyword) queries one can issue for a given time period. For example, Google's SOAP and JSON search APIs enforce a limit of 1,000 and 100 queries per user per day, respectively [2].

Query Pool: Most existing attacks for aggregate estimation use a query pool Ω which is a rich collection of queries that recall[4] most, if not all, documents in the corpus. The existing attacks [8, 9, 26] construct such a query pool by crawling an online directory, e.g., Open Directory Project [1]. Note that if a query pool cannot *recall* all documents in the corpus, then an aggregate estimation attack can only aim to analyze the subset of documents that can be recalled. Thus, for the purpose of this paper, we consider the worst-case scenario where an adversary is capable of finding a query pool that recalls all documents in the search engine's corpus.

Bipartite Graph Model: To help understand the design of aggregate estimation attacks which we shall review next, we introduce a model of bipartite graph formed by two classes of nodes corresponding to queries in the pool and documents in the corpus, respectively. An edge exists between a query node and a document node iff the query returns the document in its answer. Figure 1 depicts a simple example of such a graph. Here $k = 2$, and we only consider single-word queries in the example for the sake of simplicity. Note that dotted lines are only for illustrating "matching" relationship, and do not represent edges in the bipartite graph.

Given the bipartite graph, one can see that the sensitive aggregates are defined over nodes on the *right* side of the graph (i.e., documents), while an adversary can only "operate on" the *left* side (i.e., issue queries). As such, the main method taken by the existing aggregate estimation attacks is to study the topology of edges connecting the two sides - a topology which is determined by the query processing mechanism - and to infer the right-hand-side aggregates based on the topology. Correspondingly, our main idea of aggregate suppression, as we shall further illustrate in the paper, is to make minimum revisions to such a topology while "misleading" the adversarial estimation on the right side as much as possible.

2.2 Aggregate Estimation Attacks

We now briefly review a brute-force attack and a state-of-the-art sampling-based attack for aggregate estimation. Note that while we shall formally define the sensitive aggregates to be protected in Section 3.1, here we use COUNT(*) - i.e., the number of documents in the corpus - as the example for reviewing the attacks. Extensions to other aggregates are fairly straightforward [9].

[2]We conducted experiments using a well-known real-world query workload, the AOL query log, which is described in Section 6.1.

[3]where ARBI stands for arbitrary (queries)

[4]A document is recalled if at least one query in the pool returns it.

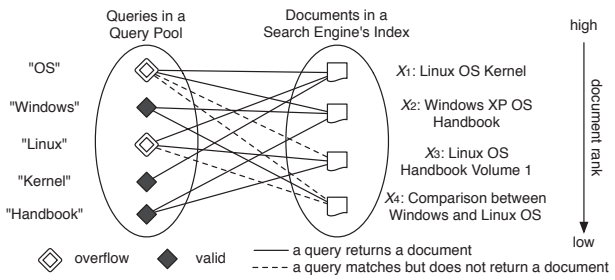

Figure 1: An Example of Bipartite Graph

Brute-Force Attack: We start with a simple brute-force attack for the purpose of illustrating how the above-described restrictions on a keyword-search interface makes aggregate estimation a subtle task. Consider a brute-force attack which exhaustively issue all queries in the query pool, with the purpose of first crawling all documents from the corpus and then generating the aggregate estimations offline. This brute-force attack does not work in practice because of the two limitations set forth by the interface on the number of documents returned by a query and the number of queries issued by a user, respectively - which essentially enforce an upper bound on the number of documents a user can crawl (e.g., from an IP address) before being blocked. Since the number of documents in a real-world enterprise search engine's corpus is usually orders of magnitude larger than this upper bound, the brute-force attack cannot crawl a large percentage of documents from the corpus.

Sampling-Based Attacks: We now review a sampling-based attack [9] which we refer to as UNBIASED-EST, in the context of estimating COUNT(*). Consider the weight $w(e)$ of each edge e in the graph as inverse of the degree of right-side node associated with the edge. In Figure 1, the weight of edge connecting "X3" and "Linux" is $1/2$ because the degree of $X3$ is 2. Then, the objective of counting the number of documents becomes estimating the SUM of all edge weights in the graph.

UNBIASED-EST does so by first selecting a query q uniformly at random from the pool Ω, and then estimating $w(e)$ for all edges associated with q. Note that $\mathrm{SUM}_{e:\langle q,\cdot\rangle}(w(e)) \cdot |\Omega|$, where $\{e : \langle q, \cdot\rangle\}$ is the set of all edges associated with q and $|\Omega|$ is the total number of queries in the pool, is a unbiased estimation of COUNT(*). UNBIASED-EST estimates $w(e)$ with a second-round sampling process: In particular, for a given edge e, one first determines the set of queries that match the right-side document X of e. Let $M(X)$ be such a set. Then, UNBIASED-EST repeatedly selects a query uniformly at random from $M(X)$ and issues it until finding one that actually returns X. Suppose that t queries (from $M(X)$) have been issued at this time. UNBIASED-EST approximates $w(e)$ by $t/|M(X)|$.

There has also been more recent work, e.g., STRATIFIED-EST [26], which improves the efficiency and accuracy of estimating $w(e)$ by applying stratified sampling techniques. While we shall test our aggregate suppression techniques against this state-of-the-art attack in Section 6, we mainly use UNBIASED-EST as the running example (of attack) throughout the paper. Again, note that the privacy guarantee achieved by our aggregate suppression techniques is generic to all possible attacks (under subtle limitations explained in Section 3.3), instead of being limited to the existing ones.

3. PROBLEM DEFINITION

3.1 Objective of Aggregate Suppression

Our objective is to suppress aggregates of the form Q_{A} : SELECT AGGR(*) FROM C WHERE *selection_condition*, where C is the search engine's corpus and *selection_condition* is a Boolean function which takes a document $X \in$ C as input and outputs whether the document is included as input to the aggregate function AGGR - e.g., if the selection condition is to whether a document's length exceeds 1,000 words and AGGR is COUNT, then the aggregate query returns the total number of documents in the corpus which has length greater than 1,000 words. In this paper, we focus on COUNT and SUM as the aggregate function. Let $Res(Q_{\mathrm{A}})$ be the answer to Q_{A}. As discussed in Section 1, due to privacy concerns the owner of an enterprise search engine may consider certain Q_{A} to be sensitive and would not willingly disclose their results. Throughout the paper, we use Q_{A}: SELECT COUNT(*) FROM D as a running example, while showing that a simple extension exists to other COUNT and SUM aggregates with or without selection conditions.

To quantify the degree of aggregate disclosure, we consider a (ϵ, δ, c)-privacy game similar in spirit to the privacy game notions in [12, 17, 19]. For a sensitive Q_{A}, consider three steps:

1. The owner applies its aggregate-suppression technique.
2. The adversary issues at most c keyword search queries and analyzes their answers to try and estimate $Res(Q_{\mathrm{A}})$.
3. The adversary wins if $\exists x$ such that the adversary has confidence $> \delta$ that $Res(Q_{\mathrm{A}}) \in [x, x + \epsilon]$. Otherwise, the defender wins.

Based on the (ϵ, δ, c)-game notion, we define the aggregate suppression guarantee for an enterprise search engine as follows:

DEFINITION 1. *We say that an aggregate-suppression technique achieves an (ϵ, δ, c, p)-guarantee if and only if for any sensitive aggregate Q_{A} and any adversary P_{A},*

$$\Pr\{P_{\mathrm{A}} \textit{ wins } (\epsilon, \delta, c)\textit{-privacy game for } Q_{\mathrm{A}}\} \leq p. \qquad (1)$$

The probability is taken over the (possible) randomness in both the aggregate-suppression scheme and the attacking strategy. One can see that the greater ϵ is or the smaller δ, c and p are, the more protection a (ϵ, δ, c, p)-privacy-guarantee has to provide on sensitive aggregates.

3.2 Objective of Utility Preservation

In addition to suppressing the sensitive aggregates, we must also maintain a high level of utility for bona fide search engine users by limiting the amount of changes to each search query answer. A document-hiding based technique like ours may introduce two types of errors: (1) *false negatives* - i.e., we might remove from a query answer a document which originally appears in the top-k list, and (2) *false positives* - i.e., we may add to the answer a document which originally ranks below the top-k results[5]. To quantify the errors, we consider two utility measures, *recall* and *precision*:

DEFINITION 2. *For a given sequence of search queries $q_1, \ldots, q_h$,*

[5]Note that a document-hiding based technique never returns a document that does not "match" the input query.

the recall *and* precision *of an aggregate suppression technique are*

$$recall = \sum_{i=1}^{h} \frac{|Res(q_i) \cap Res_{\mathrm{AS}}(q_i)|}{h \cdot |Res(q_i)|}. \quad (2)$$

$$precision = \sum_{i=1}^{h} \frac{|Res(q_i) \cap Res_{\mathrm{AS}}(q_i)|}{h \cdot |Res_{\mathrm{AS}}(q_i)|}. \quad (3)$$

where $Res(\cdot)$ and $Res_{\mathrm{AS}}(\cdot)$ are the answers to a query before and after aggregate suppression, respectively.

We shall study the recall and precision measures in both theoretical analysis and experimental studies. In addition, in experiments we shall also test *rank distance* [20] which has been extensively used to measure the quality of top-k query answers in literature.

3.3 Taxonomy of Adversaries

We consider two types of adversaries, SIMPLE-ADV and ARBI-ADV, in this paper. We start with SIMPLE-ADV which summarizes all existing aggregate estimation attacks over search engines. According to SIMPLE-ADV, an adversary possesses as prior knowledge a pool of queries. We require this query pool to satisfy two conditions: (1) it must recall all or a large percentage of documents in the corpus, and (2) for each document in the corpus, the number of queries in the pool which return it cannot exceed a small constant $d_{\max}$. The existing attacks, UNBIASED-EST and STRATIFIED-EST, satisfy both conditions. Note that the second condition is satisfied by the existing attacks for two reasons: (1) they use phrase queries of a fixed length to form the query pool, essentially limiting $d_{\max}$ to the length of a document, and (2) since the existing attacks need to compute or estimate the number of queries returning a document, they have to keep $d_{\max}$ small in order to make the computation/estimation process efficient.

To launch an aggregate estimation attack, a SIMPLE-ADV adversary chooses a query uniformly at random from query pool, retrieves all documents returned by the query, and then, for each returned document, investigates (i.e., computes or estimates) the total number of other queries which also return the document. Finally, the adversary produces an estimation based on three inputs: (1) the number of documents returned by each issued query, (2) the aggregate measure[6] over each retrieved document, and (3) the number of other queries which return each retrieved document. A formal model of SIMPLE-ADV will be introduced in Section 4.1. One can see that both existing attacks - i.e., UNBIASED-EST and STRATIFIED-EST - follow this SIMPLE-ADV model[7].

In addition to SIMPLE-ADV, we also consider ARBI-ADV which allows the adversary to issue an arbitrary set of queries and then produce an estimation based on the same three inputs as described above for SIMPLE-ADV. Note that the above-described two limitations on the query pool no longer apply, and the adversary does not necessarily select queries uniformly at random from the query pool.

3.4 Problem Statement

Our objective is to achieve $(\varepsilon, \delta, c, p)$-privacy guarantee against SIMPLE-ADV (resp. ARBI-ADV) adversaries while maximizing recall and precision for a bona fide workload of search queries.

[6]e.g., 1 if COUNT is the aggregate function, the document length if SUM(doc_length) is the aggregate function.

[7]Note that while UNBIASED-EST requires one query pool that recalls almost all documents in the corpus, STRATIFIED-EST uses a small number of disjoint query pools, each representing a stratum. Nonetheless, both algorithms follow the SIMPLE-ADV model by choosing queries uniformly at random from each query pool.

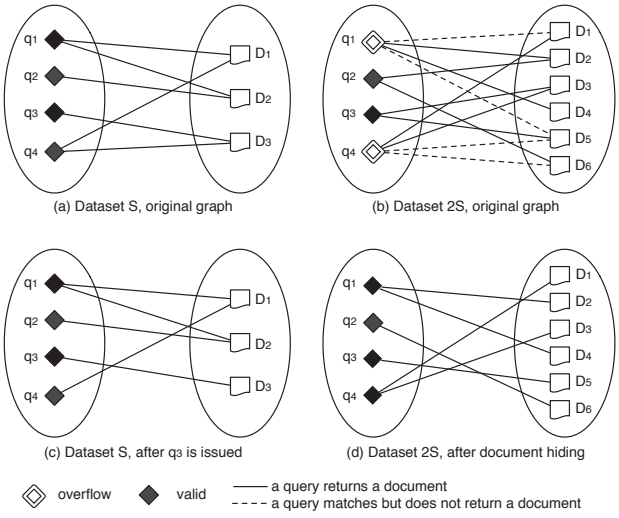

Figure 2: Running Example

4. AS-SIMPLE

In this section, we describe our main techniques for thwarting aggregate estimation attacks launched by a SIMPLE-ADV. We shall first develop the basic ideas of our defense using a two-corpora example, and then present the generic AS-SIMPLE algorithm. We shall also demonstrate a case study of how AS-SIMPLE thwarts the existing UNBIASED-EST attack, and derive the privacy and utility guarantees AS-SIMPLE achieves.

4.1 Basic Ideas of AS-SIMPLE

For the ease of understanding, we start with considering an adversary that aims to estimate the size of a search engine's corpus. Consider a running example depicted in Figure 2, where corpora S and $2S$ differ twice in their sizes. One can see that, if an adversary is capable of accurately estimating a search engine's corpus size, it must be able to distinguish S and $2S$ based on the search query answers it receives from the two search engines. Thus, we consider a SIMPLE-ADV's ability of distinguishing between S and $2S$ as a running example throughout this section. In particular, we start with discussing how an adversary can make such a distinction, and then describe our basic ideas for document hiding which makes S and $2S$ *indistinguishable* from the view of a SIMPLE-ADV, unless the adversary issues an extremely large number of queries. One can see that this essentially thwarts size estimation attacks by SIMPLE-ADV over the two corpora.

How Attack Works: Recall from our taxonomy of adversaries in Section 3.3 that, for size (i.e., COUNT) estimation, a SIMPLE-ADV relies on two inputs: (1) the number of documents returned by each query the adversary issues, and (2) the number of queries which returns each document the adversary retrieves. With the bipartite graph model depicted in Figure 2, the two inputs are corresponding to the degree of a left-hand-side (LHS) node (i.e., a query) and a right-hand-side (RHS) node (i.e., a document), respectively.

Intuitively, one can see that the larger the LHS degrees are, the higher the COUNT estimation should be because more documents are retrievable by queries in the pool. On the other hand, the higher the RHS degrees are, the lower the COUNT estimation should be because more overlapping occurs between documents returned by different queries. For the example of S and $2S$, if no overflow happens on the search queries (i.e., when k is sufficiently large),

the LHS degree over $2S$ is (expectedly[8]) twice as much as that over S, while their RHS degrees are (expectedly) the same - making it easy for a SIMPLE-ADV to distinguish between the two corpora based on their different LHS degrees[9].

Key Challenges for Defense: Recall from the introduction section a seemingly simple idea to thwart SIMPLE-ADV from making the distinction: for each query q received by the search engine, instead of applying top-k filtering over all $|q|$ documents matching the query, one first selects $|q|/2$ documents from the matching ones (and "hides" the other half), and then applies top-k filtering over the $|q|/2$ documents to produce the final query answer $R(q)$. After applying this idea over the running example, the bipartite graphs over S and $2S$ are depicted in Figure 2(a) and (d), respectively. One can see that this idea equalizes the LHS degrees over both S and $2S$. Nonetheless, as we explained in Section 1, it fails on aggregate suppression because, by reducing the LHS degree over the $2S$ corpus, this approach also reduces the RHS degrees of the $2S$ documents (by half when no overflow occurs - as one can observe from Figure 2(a) and (d)) - enabling adversarial distinction based on the RHS degrees.

An observation from the failure of this simple approach is that, to truly make S and $2S$ indistinguishable from each other, a aggregate suppression technique must equalize both LHS and RHS degrees over the two corpora. Unfortunately, this requirement brings the key challenge for defense - as it is impossible to do so perfectly without significantly sacrificing the utility of search engines. To understand why, note that the number of LHS nodes - i.e., the number of queries in the pool - are constant over any corpus. Thus, equalizing LHS degrees means equalizing the number of edges, which in turn leads to equal number of RHS nodes (i.e., documents) when RHS degrees are equal. In other words, the only way to perfectly equalize both LHS and RHS degrees is to completely removes S documents from the $2S$ corpus - an approach which comes with significant utility loss, as we argued in Section 1.

Given the inherent difficulty of perfectly equalizing LHS and RHS degrees, the design objective of a document hiding technique should be to *delay* a SIMPLE-ADV from making the distinction until it issues an extremely large number of queries. We describe the main idea of AS-SIMPLE for doing so as follows. Note that in the description, without causing ambiguity, we use term *edges* and *degrees* to refer to *matching* relationships between a query and a document - i.e., both solid and dotted lines in Figure 2.

Main Idea of AS-SIMPLE: Unlike the above-described simple approach, AS-SIMPLE performs document hiding over *both* S and $2S$ corpora. For the $2S$ corpus, AS-SIMPLE employs the same procedure as the simple approach - i.e., removing half of all edges connected to a query. Note that one may select the removed edges as the ones corresponding to the lower-ranked documents, in order to preserve utility for bona fide users. Figure 2(d) shows for the running example the bipartite graph after document hiding over $2S$. For the S corpus, AS-SIMPLE removes half of all edges connected to a document (i.e., hiding the document from half of all matching queries) after the document is returned for the first time.

Unlike in the $2S$ case, edge removal over S cannot be done deterministically at once because the defender does not have knowledge of the query pool used by the attacker (and therefore does not know which edges to select from for removal - or even how many edges to remove). Instead, AS-SIMPLE performs edge removal over S in an online fashion. In particular, since the second time a document is returned, there is a 50% chance for the document to be removed from the query answer (and replaced with a lower-ranked document if the query overflows). One can see that this online approach produces an equivalent bipartite graph as the original description. For the running example, Figure 2(c) depicts the bipartite graph once document hiding is performed after q_3 is processed.

We now first discuss the motivation for AS-SIMPLE to add document hiding (i.e., edge removal) over S, and then explain why it effectively delays the adversarial distinction of S and $2S$. The main purpose for performing edge removal over S is to avoid the pitfall of the simple approach - in particular, one can see that the edge removal process reduces the RHS degrees over S to the same level as $2S$ after document hiding (e.g., for D_3 over Figures 2(c) and (d) after q_3 is issued). This prevents an adversary from making the distinction based on RHS degrees.

Nonetheless, it is equally important to note that this edge removal process *cannot* prevent the distinction forever. To understand why, note that the removal of edges over S reduces not only the RHS degrees of documents but also the LHS degrees of many queries as well. Since AS-SIMPLE performs edge removals over S after a document is returned for the first time, if an adversary tracks the LHS degrees of S and $2S$ over time, it may discover that, while the LHS degrees over $2S$ remains (expectedly) constant, the LHS degrees over S keeps decreasing over time - enabling the adversarial distinction between S and $2S$. Again, this distinction is ultimately inevitable because, when an adversary issues enough queries to crawl the two corpora, it will certainly detect either smaller LHS degrees for S, or smaller RHS ones for $2S$. Nonetheless, a key question remains - how many queries will a SIMPLE-ADV have to issue before reaching distinction.

Fortunately, our analysis shows that a SIMPLE-ADV needs to issue an extremely large number of queries before making the distinction. While the formal results will be presented in Section 4.4, we provide an intuitive explanation here as follows. Consider how an adversary can identify the LHS degrees of S being smaller. Note that any query which has a substantially smaller degree over S than $2S$ (after document hiding) must return a significant number of documents that the adversary has retrieved before. Nevertheless, since the query pool for SIMPLE-ADV is built such that each document is only returned by a small number of ($\leq d_{\max}$) queries, and a SIMPLE-ADV chooses queries uniformly at random from the pool, it needs to retrieve an expected number $O(\sqrt{|S|/d_{\max}})$ unique documents from S before hitting one document that has been previously retrieved, where $|S|$ is the number of documents in S and satisfies $|S| \gg k$ and $|S| \gg d_{\max}$ in practice. In addition, one can see that an adversary cannot identify the LHS degrees of S being lower based on just one repetitive document - instead, a significant number is needed given the inherent variance of LHS degrees for different queries. Thus, after AS-SIMPLE is deployed, a SIMPLE-ADV needs to issue a very large number of queries before making the distinction.

4.2 Algorithm AS-SIMPLE

We now describe how to generalize the basic idea of AS-SIMPLE to a practical algorithm in three steps: First, we show how it can be used to obfuscate two corpora with COUNT differing by γ times, where γ is the *obfuscation factor* measuring how stringent the aggregate suppression guarantee is - in particular, the larger γ is, the more stringent the suppression guarantee will be. We shall also explain in this part how our idea can obfuscate any corpus size falling in the range of $[|S|, \gamma \cdot |S|]$. Second, we note that our idea de-

[8]Recall that the corpus S is sampled without replacement from $2S$. The expected values mentioned in this paragraph are taken over the randomness of this sampling process.

[9]With a smaller k (and therefore overflowing queries), the LHS and RHS degrees over both corpora may decrease because of the overflows, with the decrease over $2S$ being slightly higher, as more queries are likely to overflow over the larger corpus.

Algorithm 1 AS-SIMPLE

Input: Corpus size n, obfuscation factor γ
$\Theta_{\mathrm{R}} \leftarrow \phi$. *//set of documents returned before*
Wait for input query q.
Compute $\mu \leftarrow n/\gamma^{\lfloor \log n / \log \gamma \rfloor}$.
$M(q) \leftarrow$ the set of $\min(|q|, \gamma \cdot k)$ highest ranked documents matching q.
$Res(q) \leftarrow M(q)$.
for each document D in $Res(q)$ **do**
  if $D \in \Theta_{\mathrm{R}}$ **then**
    with $1 - \mu/\gamma$ probability, remove D from $Res(q)$.
  else
    $\Theta_{\mathrm{R}} = \Theta_{\mathrm{R}} \cup \{D\}$.
  end if
end for
Remove $\max(|Res(q)| - 1/\mu \cdot |M(q)|, |Res(q)| - k)$ lowest-ranked documents from $Res(q)$.
return $Res(q)$ as the answer to q. Goto 3

scribed above calls for different document hiding procedures over S and $2S$. Since a real-world search engine only has one corpus and thus one size, we explain here how a defender can determine which procedure to apply. Finally, we explain why the suppression of COUNT(*) also leads to the suppression of answers to other COUNT and SUM queries with or without selection conditions. Algorithm 1 depicts the generalized AS-SIMPLE algorithm.

Arbitrary Obfuscation Factor γ: Observe from the basic idea that the main procedure followed by AS-SIMPLE on manipulating LHS and RHS degrees can be summarized as follows: For the larger corpus (e.g., $2S$), AS-SIMPLE reduces its LHS degrees to the same level as those over the smaller corpus. For the smaller corpus (e.g., S), AS-SIMPLE reduces its RHS degrees to the same level as those over the larger corpus (after document hiding). The same procedure can be followed to obfuscate all corpora of sizes falling in the range of $[|S|, \gamma \cdot |S|]$ for an arbitrary γ. In particular, given a corpus of size $\mu \cdot |S|$ ($\mu \in [1, \gamma]$), AS-SIMPLE first reduces its RHS degrees to the same as those over γS - by removing $(1 - \mu/\gamma)$ of all edges associated with each document. Then, AS-SIMPLE reduces its LHS degrees to the same as those over S - by removing $(1 - 1/\mu)$ of all edges associated with each query. One can see that, when $\gamma = 2$, this procedure is reduced to the above-described ones for S and $2S$, respectively, when $\mu = 1$ and 2. Another observation is that the larger γ is, the less utility will be preserved for bona fide search users (as a tradeoff for the more stringent suppression guarantee).

Indistinguishable Segments: For a given corpus, AS-SIMPLE first finds a pre-determined *indistinguishable segment* it belongs to based on the size of the corpus, and then apply document hiding according to the segment. In particular, given a user-input obfuscation factor γ, we consider indistinguishable segments $[1, \gamma]$, $[\gamma, \gamma^2], \ldots, [\gamma^i, \gamma^{i+1}], \ldots$ As such, a corpus of size n will fall into the $\lceil \log n / \log \gamma \rceil$-th segment. Then, we apply document hiding according to the procedure outlined above for obfuscating this segment.

Other COUNT and SUM queries: One can see that the suppression of COUNT(*) also extends to other COUNT and SUM queries with or without selection conditions, because by emulating a corpus of size γ times as large, AS-SIMPLE simultaneously "enlarges" all other COUNT and SUM aggregates for the same (expected) ratio. We formally derive the suppression guarantee provided by AS-SIMPLE at the end of this section.

4.3 Case Study Against UNBIASED-EST

We now conduct a case study of AS-SIMPLE against an existing aggregate estimation attack. For the sake of simplicity, we choose UNBIASED-EST for the case study. Note that our AS-SIMPLE algorithm is not designed specifically for defending against this attack. It effectively thwarts a broad class of SIMPLE-ADV adversaries, as we shall prove in the suppression guarantee at the end of this section. In addition, we shall also demonstrate in the experimental results that AS-SIMPLE can also effectively thwart the other existing attack, STRATIFIED-EST.

Consider again the running example depicted in Figure 2 and an adversary which runs UNBIASED-EST to estimate the total number of documents in the corpora. Suppose that the first query randomly chosen by the adversary is q_3. We first follow the procedure reviewed in Section 2.2 for UNBIASED-EST to compute the adversarial estimations from q_3 before AS-SIMPLE is applied. For S, the adversary retrieves 1 document D_3 from q_3. Since D_3 has a degree of 2, the edge $\langle q_3, D_3 \rangle$ is assigned a weight of $w = 1/2$. Given that the query pool contains $|\Omega| = 4$ queries, the adversarial estimation becomes $|S| \approx |\Omega| \cdot w = 4 \times 1/2 = 2$. Similarly, over $2S$, q_3 returns two documents: D_3 with (return) degree of 2 and D_5 with (return) degree of 1. The total weight of edges associated with q_3 then becomes $w = 1/2 + 1 = 3/2$. Thus, the adversarial estimation becomes $|2S| \approx |\Omega| \cdot w = 6$. One can see that the estimation over $2S$ is twice as much as that over S, clearly indicating the adversary's ability to make the distinction.

Now compute the estimation generated by UNBIASED-EST after AS-SIMPLE is applied. We consider $2S$ first. After document hiding, q_3 only returns one document D_5, which has a degree of 1. Thus, the adversarial estimation is $|2S| \approx 4 \times 1 = 4$. For S, note that since q_3 is the first query being issued, the topology of the bipartite graph remains unchanged (i.e., no removed edges) when q_3 is being processed. Thus, q_3 retrieves one document D_3. Nonetheless, immediately afterwards, half of all edges associated with D_3 are removed - e.g., as one can see from the figure, edge $\langle q_4, D_3 \rangle$ is removed in the example. As such, when the adversary now computes or estimates the degree of D_3, the result it obtains is 1. In turn, the adversarial estimation becomes $|S| \approx 4 \times 1 = 4$, exactly the same as the estimation over $2S$, indicating the adversary's inability to distinguish between the two corpora.

While we do not repeat the case study for all other queries, we do want to point out that, as one can verify using the topology of $2S$ after document hiding, UNBIASED-EST indeed remains unbiased (and accurate) over $2S$ even after AS-SIMPLE is applied[10]. The effectiveness of AS-SIMPLE on thwarting UNBIASED-EST is actually enabled by its ability to "disguise" UNBIASED-EST into overestimating COUNT(*) on S. While a small example like Figure 2 is not the best way to demonstrate the power of disguise because soon the adversary can crawl all documents, one can actually observe the source of overestimation from the above-described example of issuing q_3. Note that when the adversary issues q_3, according to the topology of the bipartite graph as depicted in Figure 2(a), edge $\langle q_3, D_3 \rangle$ has a weight of $1/2$. Nonetheless, when the adversary winds up estimating this weight, it has become 1 because of the edge removals performed by AS-SIMPLE. This leads to a twice-overestimation by UNBIASED-EST over S, and prevents the adversary from making the distinction between S and $2S$ until issuing a large number of queries (over a much larger query pool and corpus than depicted in the example, of course).

[10]The estimations from $q_1, \ldots, q_4$ are $8, 4, 4, 8$, respectively, leadings to an unbiased expected value of 6, the real COUNT(*).

4.4 Performance Analysis

Privacy Guarantees: We now derive a spectrum of $\langle \epsilon, \delta, c, p \rangle$-aggregate suppression guarantees AS-SIMPLE achieves against the broad class of SIMPLE-ADV adversaries.

THEOREM 4.1. *Given a search engine with corpus of size n and a top-k interface, for any sensitive COUNT or SUM aggregate with value q_{A} and any $\delta \in [0, 1]$, AS-SIMPLE with an amplification factor γ achieves $\langle \gamma^{\lceil \log n / \log \gamma \rceil} \cdot \delta \cdot q_{\mathrm{A}}/n,\ \delta,\ \sqrt{n/(d_{\max} \cdot k)},\ 50\% \rangle$ aggregate suppression guarantee against any SIMPLE-ADV adversary which has its query pool having a maximum RHS degree of $d_{\max}$.*

Due to the space limitation, we do not include theorem proofs in the paper. The theorem extends our above-mentioned intuition to show that, for a real-world enterprise search engine, the adversary needs to issue a very large number of queries in order to obtain an accurate estimate of SUM and COUNT aggregates - this effectively thwarts the aggregate estimation attacks from SIMPLE-ADV.

Utility for Bona Fide Search Users: We now consider the utility provided by AS-SIMPLE to bona fide search users according to *recall* and *precision*, the two utility measures defined in Section 3.2.

THEOREM 4.2. *Given an amplification factor γ and a workload[11] of bona-fide search queries Ω_{B} which returns an average of d documents, and includes $\rho_\gamma \cdot |\Omega_{\mathrm{B}}|$ queries that match more than $\gamma \cdot k$ documents as well as $\rho_{\mathrm{O}} \cdot |\Omega_{\mathrm{B}}|$ queries that overflow ($\rho_{\mathrm{O}} \geq \rho_\gamma$), AS-SIMPLE achieves*

$$recall \geq \min\left[\frac{\rho_\gamma \cdot (\gamma - 1) + 1}{\gamma}, \frac{d \cdot |\Omega_{\mathrm{B}}| + (\gamma - 1) \cdot n_1}{\gamma \cdot d \cdot |\Omega_{\mathrm{B}}|}\right] \quad (4)$$

$$precision \geq 1 - \left(1 - \frac{1}{\gamma}\right) \cdot \rho_{\mathrm{O}}. \quad (5)$$

over any search engine's corpus where n_1 is the number of documents returned exactly once by the query workload.

One can see from the theorem that, consistent with our intuition on the tradeoff between utility and aggregate suppression, the larger the amplification factor γ is, the worse the utility of AS-SIMPLE will be - indicating a tradeoff between aggregate suppression and search utility.

5. AS-ARBI

In this section, we consider a broader set of ARBI-ADV adversaries which may construct an arbitrary query pool and issue query from them according to arbitrary distribution. In particular, we shall first describe an attack against AS-SIMPLE using a query pool consisting of *highly correlated* queries - i.e., queries which return largely overlapping documents. Then, we shall present the main idea of AS-ARBI and explain how it addresses the correlated-query based attack.

5.1 An Attack Against AS-SIMPLE

Before describing the attack, we first recall from the above section the main reason why AS-SIMPLE cannot thwart the aggregate estimation attacks forever - because of the document hiding procedure taken by AS-SIMPLE, the more queries an adversary issues, the lower the LHS degrees will be over S. Eventually, after an adversary issues a large number of queries, the LHS degrees over S would be substantially lower than those over $2S$ - enabling the adversarial distinction between the two corpora. The main idea of the attack is to push this reduction of LHS degrees over S to the extreme - in particular, it uses a specially designed query pool to rapidly reduce the LHS degrees over S when an adversary issues only a small number of queries.

[11]Note that the workload may contain duplicate queries.

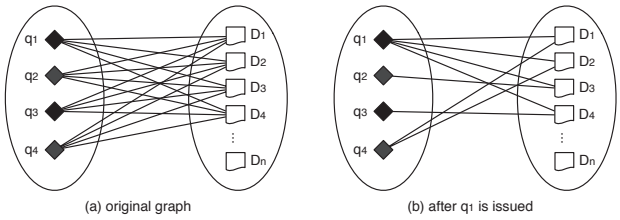

Figure 3: Example of Attack

In particular, Figure 3(a) depicts an example of the specially designed query pool. The key property here is that the queries in the pool are strongly correlated - i.e., the documents returned by all queries in the pool significantly overlap with each other. The motivation for having this property is explained in Figure 3(b), which depicts the bipartite graph after q_1 is issued when the corpus falls on the lower end of an indistinguishable segment (i.e., serve as S in the two-corpora case). Specifically, one can see that the overlapping query answers make the LHS degrees of all queries in the pool to decrease significantly after the processing of just one query (q_1). As a result, the adversary can quickly distinguish this corpus from one γ times as large - disclosing aggregate COUNT(*).

It is important to note that this attack by no means contradicts our aggregate suppression guarantees derived in Section 4, because the query pool used in the attack clearly violates our definition of SIMPLE-ADV. In particular, the SIMPLE-ADV model requires (1) the queries to recall a large percentage of documents in the query pool, and (2) the adversary to issue queries chosen uniformly at random from the pool. Thus, even if $q_1, \ldots, q_4$ in Figure 3 are in a query pool constructed by a SIMPLE-ADV, they would only form a rather small portion of the pool. As such, it is highly unlikely for an adversary to issue multiple queries in $q_1, \ldots, q_4$, and thereby make a successful attack, before issuing a large number of queries.

Nonetheless, we would also like to remark that, as we shall demonstrate in the experiments section, it is indeed possible to construct queries like $q_1, \ldots, q_4$ and launch such a correlated-query based attack in practice. Specifically, while the adversary cannot precisely determine the precise overlapping of query results before actually issuing these queries to the search engine, it is possible for the adversary to infer the potential overlaps by analyzing an external (linguistic) corpus of documents. Our tests in the experiments section show that an adversary can quickly observe the reduction of LHS degrees by issuing these correlated queries.

5.2 Basic Idea of AS-ARBI

Before introducing the main idea of AS-ARBI, we start with describing a simple yet effective method for defending against the attack. Since the attack aims to use a small number of queries to repeatedly cover a small number of documents, a simple idea for defense is to *decline* (i.e., refuse to answer) a query if the vast majority of documents it matches can be "covered" by a small number of historic queries - i.e., for a query q, there exists at most m historic queries $q_1, \ldots, q_m$ such that

$$|q \cap (Res(q_1) \cup \cdots \cup Res(q_m))| \geq \sigma \cdot |q|, \quad (6)$$

where q and $Res(q_i)$ represent the set of documents matched and returned by a corresponding query, respectively, and both *cover size*

m and *cover ratio* σ are pre-determined parameters (we shall explain how to find $q_1, \ldots, q_m$ in the next subsection). One can see that, as long as we decline q over all corpora in an indistinguishable segment (i.e., over both S and $2S$ in the two-corpora example), the adversary cannot infer from the decline response whether there has been a significant decrease of LHS degrees (because the response would be the same even if the corpus is $2S$ and therefore has no reduction of LHS degrees) - effectively thwarting the correlated-query based attack.

Nonetheless, this simple idea comes at a major cost on utility. In particular, note that different bona fide search users may indeed issue similar yet different queries - e.g., "SIGMOD 2012", "2012 SIGMOD", "ACM SIGMOD 2012", etc - which retrieve significantly overlapping search results. Declining such queries will lead to a significant drop on recall (to 0 for the declined query). To address this deficiency, AS-ARBI features the idea of *virtual query processing* which we describe as follows.

To understand how virtual query processing works, note that if the adversary issued $q_1, \ldots, q_m$, it already has query answers $Res(q_1), \ldots, Res(q_m)$. Thus, even if the new query q is declined by the search engine, the adversary can still figure out an answer by itself through searching for documents which have been returned in previously issued queries and match q.

An interesting observation here is that, since the search engine does not disclose any information by declining the query, the adversary will not learn anything extra from the self-constructed query answer - i.e., it is safe for the search engine to provide the same query answer to every user without violating the aggregate suppression guarantee. As such, with virtual query processing, once a query is declined, the search engine then processes the query "virtually" by composing an answer from the historic ones. One can see that doing so significantly increases utility (specifically, recall) without affecting aggregate suppression. We shall discuss the details of virtual query processing in the next subsection.

5.3 Algorithm AS-ARBI

Algorithm 2 AS-ARBI

1: **Input:** Cover size m, cover ratio σ
2: $\Omega_{\mathrm{H}} \leftarrow \phi$. *//historic query set*
3: Wait for input query q.
4: **if** $\exists\{q_1, \ldots, q_u\} \subseteq \Omega_{\mathrm{H}}$ $(u < m)$ such that $|q \cap (Res(q_1) \cup \cdots \cup Res(q_u))| \geq \sigma \cdot |q|$ **then**
5: Return $|q \cap (Res(q_1) \cup \cdots \cup Res(q_u))|$ (subject to top-k filtering) as the answer to q.
6: **else**
7: Call AS-SIMPLE to answer q. $\Omega_{\mathrm{H}} \leftarrow \Omega_{\mathrm{H}} \cup \{q\}$.
8: **end if**
9: Goto 3

Algorithm 2 depicts the AS-ARBI algorithm. One can see that it essentially serves as a pre-processing step (of virtual query processing) for AS-SIMPLE. We now discuss an implementation issue for AS-ARBI on finding whether documents matching the current query can be covered by (at most) m historic query answers - i.e., whether virtual query processing should be triggered.

First, we would like to note that the percentage of real search queries triggering virtual query processing is usually small because the trigger allows query q to match at most $k \cdot m$ documents - disqualifying a large number of broad (and overflowing) queries issued by real search users. To further reduce the overhead of evaluating the trigger, we maintain one string array and one binary vector for each document. The string array stores all queries which have returned the document. The binary vector is set to $1,000$ bits in our experiments with initial values of 0. For each query which returns the document, we hash the query string to a value in $[1, 1000]$, and then set the corresponding bit of the binary vector to 1.

Given the binary vectors, in order to determine whether a query q satisfies the trigger, we first retrieve the $|q|$ vectors corresponding to the documents which match q. We then compute the SUM of these vectors to produce a size-1,000 integer vector. After that, we find the m largest values in the vector, and determine if the SUM of these values exceeds $\sigma \cdot |q|$. If not, one can see that q definitely does not trigger virtual query processing. If the SUM does exceed $\sigma \cdot |q|$, we then enumerate all size-m combinations of queries stored in the string arrays to evaluate the trigger. Note that we choose to perform simple enumeration instead of the approximate algorithms for MIN-SET-COVER because, through experimental studies, we found the input size to be very small in practice. In particular, we shall show in practice that this trigger evaluation process incurs minimum overhead even when the historic query set Ω_{H} contains a large number of queries.

Before concluding our discussions of AS-ARBI, we would like to note a limitation of it - our ARBI-ADV model does *not* capture all possible adversaries with arbitrary external knowledge - specifically, because we limit what an ARBI-ADV can use for aggregate estimation to the three inputs described in Section 3.3. For example, consider an adversary which aims to compare the corpora size of two news agencies. Suppose that one corpus C_1 is actually a subset of the other C_2. Suppose that the adversary (somehow) learns from external knowledge all documents in $C_2 \backslash C_1$. It is easy to see that no effective defense is possible (other than actually shrinking C_2 to C_1) because the adversary can always test a document in $C_2 \backslash C_1$ can be returned by a search engine. This adversary is not captured by our ARBI-ADV (and therefore not addressed in this paper), because it uses information beyond the three inputs, in particular the *contents* of returned documents, to make the aggregate estimation. While we note the difficulty of dealing with such adversaries, we leave as an open problem whether there exists any "reasonable" aggregate estimation attack beyond ARBI-ADV, and how to defend against such attacks if there does exist one.

6. EXPERIMENTS

We present our experimental results in this section. In particular, we start with introducing the experimental setup, and then describe our test results for AS-ARBI and AS-SIMPLE against all existing aggregate estimation attacks over various corpora of real-world documents and real-world log of bona fide search queries.

6.1 Experimental Setup

Hardware and Platform: All of our experiments were performed on two computers with Intel Core 2 Duo 2.4GHz CPU, 4GB RAM and 32bit Windows 7 operating system. All algorithms were implemented in C#.

Enterprise Search Engine: In all experiments, we used the preloaded Windows Search 4.0 in Windows 7 as the search engine, and used its default ranking function. For the top-k interface, we set $k = 5$ by default, and also tested cases when $k = 50$.

Document Corpus: We followed the technique used in the prior work for aggregate estimation attacks [8, 9, 26] to collect the document corpora used in this paper. Specifically, we first crawled 1 million web pages from the ODP online directory [1]. Then, we removed web pages with format other than htm, html and txt, webpages which contain fewer than 10 words (because most of them are HTTP 404 error pages), as well as non-English webpages. Finally,

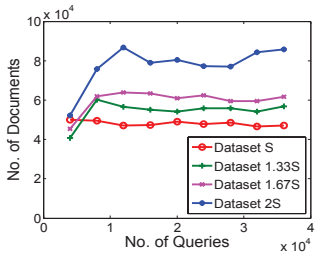

Figure 4: UNBIASED-EST: # docs vs # queries

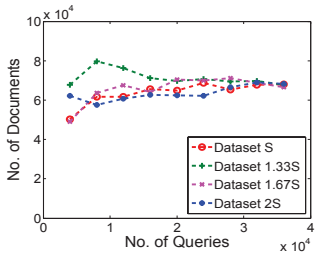

Figure 5: AS-ARBI: # docs vs # queries

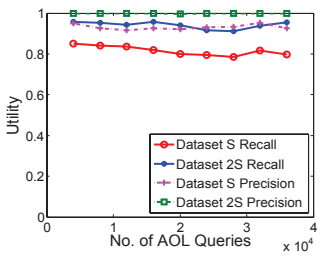

Figure 6: AS-ARBI: recall&precision vs # queries

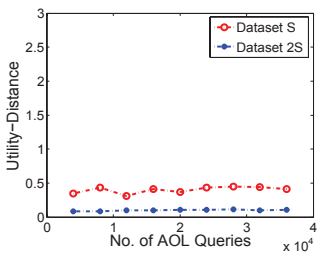

Figure 7: AS-ARBI: dis vs # queries

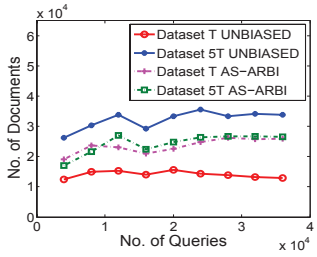

Figure 8: UNBIASED-EST&AS-ARBI: # docs vs # queries

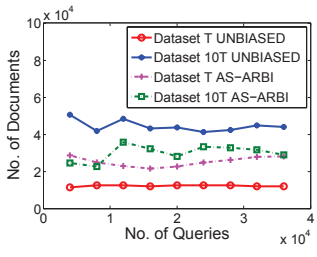

Figure 9: UNBIASED-EST&AS-ARBI: # docs vs # queries

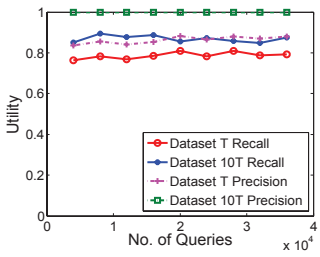

Figure 10: AS-ARBI: recall&precision vs # queries

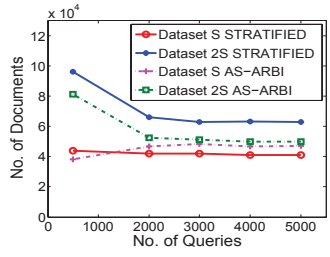

Figure 11: STRATIFIED-EST&AS-ARBI: # docs vs # queries

we sampled without replacement 150,000 documents from the result to form the corpus for our experiments. In order to test corpora of various sizes, we further sampled without replacement corpora with sizes ranging from 1,000 to 125,250 from the 150,000-document corpus.

Adversarial Query Pool: We constructed the adversarial query pool according to the existing work on aggregate estimation attacks [9, 26]. In particular, from the documents *not* chosen into the 150,000-document corpus, we sampled without replacement 50,000 documents, and extracted a total number of 135,133 different words from them to form the query pool. We chose to use single-word query pool in this paper mainly because it was demonstrated in [26] that a single-word query pool enables efficient yet accurate aggregate estimations.

Bona Fide Search Query: To test the effects of our algorithms on preserving the utility of search engine for real search users, we evaluate the utility with a real-world search engine query log - the AOL query log released by AOL Research in 2006 [3]. The log contains contains queries issued by 650,000 users over a 3-month period. In our experiments, we used the first 35,000 queries in the AOL query log, and issued them consecutively to form our workload of bona fide search queries.

Aggregate Estimation Attacks: We tested two aggregate estimation attacks, UNBIASED-EST [9] and STRATIFIED-EST [26], in this paper. UNBIASED-EST is parameter-free (besides the above-described query pool). STRATIFIED-ESTIMATOR has two parameters for estimating COUNT(*): the number of strata and the number of pilot queries (per stratum). We conducted experiments with default settings specified in [26]. In particular, we set the number of strata to 10, with the same strata design as that specified in [26]. Also according to [26], we set the number of pilot queries to 5 per stratum.

Aggregate Suppression Algorithms: We tested both AS-SIMPLE and AS-ARBI proposed in this paper. AS-SIMPLE has only one parameter: the obfuscation factor γ. For most experiments, we set γ to its default value of 2. Nonetheless, we also tested cases where $\gamma = 5$ and 10. AS-ARBI has two additional parameters, the cover size m and the cover ratio σ. We set $m = 5$ and $\sigma = 100\%$ as the default value. Note that we set σ to the most "conservative" value because we found through experiments that such a setting already addresses the correlated-query based attack and enables a high utility (specifically, recall). We also tested with values of m varying from 1 to 10, and found it bears little difference on the performance of AS-ARBI in terms of aggregate suppression, recall, precision and processing overhead.

6.2 Result Evaluation

Aggregate Suppression by AS-ARBI: We started by testing the effectiveness of AS-ARBI on suppressing aggregates, in particular COUNT(*) over the document size. To do so, we ran UNBIASED-EST over four corpora S, $1.33S$, $1.67S$ and $2S$ with 47,722, 57,855, 70,829, and 72,310 documents[12], respectively, which can be recalled by the adversarial query pool. One can see from Figure 4 that, before AS-ARBI is applied, an adversary can easily distinguish between the four corpora based on the estimations generated by UNBIASED-EST. Figure 5 depicts the estimations produced by UNBIASED-EST after AS-ARBI is applied. One can see that an adversary can no longer distinguish between the four corpora based on the estimations - demonstrating the effectiveness of AS-ARBI on thwarting the UNBIASED-EST aggregate estimation attack.

Utility for AS-ARBI: We then tested how well AS-ARBI preserves the utility of search engine for bona fide search users, using the AOL query pool described in Section 6.1. Figure 6 depicts the recall and precision while Figure 7 shows the rank distance [20] after AS-ARBI is applied. One can see that the search utility is well preserved by AS-ARBI - with recall above 80%, precision above

[12]Note that they do not exactly follow the 1:1.33:1.67:2 ratio because they yield different recalls over the adversarial query pool.

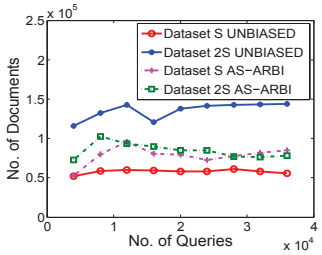

Figure 12: UNBIASED-EST&AS-ARBI(k=50): # docs vs # queries

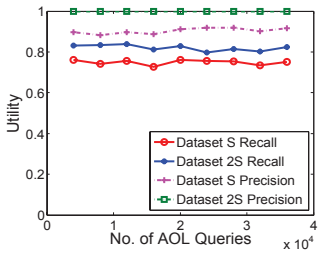

Figure 13: AS-ARBI(k=50): recall&precision vs # queries

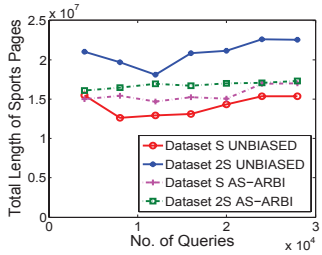

Figure 14: UNBIASED-EST&AS-ARBI: length of total sports pages vs # queries

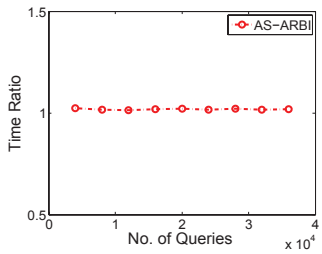

Figure 15: # time ratio vs # queries

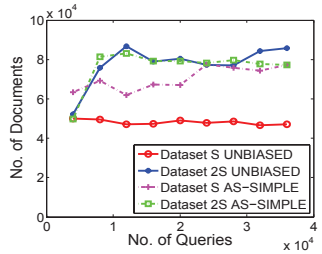

Figure 16: UNBIASED-EST&AS-SIMPLE: # docs vs # queries

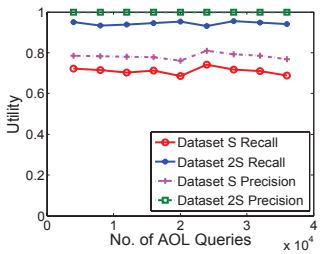

Figure 17: AS-SIMPLE: recall&precision vs # queries

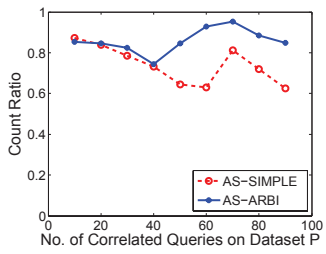

Figure 18: # count ratio vs # correlated queries

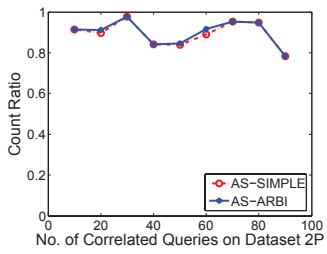

Figure 19: # count ratio vs # correlated queries

90%, and average rank distance below 0.5, under almost all settings being tested.

Various Amplification Factor γ: Next, we tested AS-ARBI with varying amplification factor. Three corpora are used: Corpus T (10,000 documents, adversary recall 85.11%), $5T$ (50,000 documents, adversary recall 67.10%), and $10T$ (100,000 documents, adversary recall 57.97%). Figures 8 and 9 depict the estimation results of UNBIASED-EST with and without AS-ARBI when the amplification factor is 5 and 10, respectively. One can see from both figures that AS-ARBI successfully suppressed COUNT(*) with the larger amplification factors. One may notice that though corpus $10T$ is ten times larger than corpus T, estimation of UNBIASED-EST for COUNT(*) on $10T$ is not that large. The reason is that *recall* of the adversarial query pool on corpus $10T$ is much smaller than that on corpus T, as we mentioned above. AS-ARBI is not affected by the decrease of recall because it has no knowledge of the adversarial query pool.

Figure 10 depicts the recall and precision for corpus T and $10T$ after AS-ARBI is applied with an amplification factor $\gamma = 10$. We also tested the utility of T and $5T$ when $\gamma = 5$, and found similar results. One can make two observations from the figure: (1) compared with the case with a smaller amplification factor $\gamma = 2$, here the recall and precision decreases slightly, indicating the tradeoff between aggregate suppression and utility preservation. (2) despite of the slight decrease, utility is still well preserved, with recall above 75% and precision above 80% for all tested settings.

Aggregate Suppression Against STRATIFIED-EST: We tested AS-ARBI against STRATIFIED-EST, the other existing aggregate estimation attack. One can see from Figure 11 that AS-ARBI effectively prevents STRATIFIED-EST from accurately estimating COUNT(*). Note that the utility results (over AOL query log) remains the same as Figures 6 and 7, because AS-ARBI is not aware of, and does not change with, the attacking algorithm.

Larger k: Figures 12 and 13 depict the effectiveness of AS-ARBI on suppressing COUNT(*) and preserving utility when $k = 50$ respectively. Note that with the larger k, the adversarial recall now becomes 88.47% for S and 80.47% for $2S$. One can see from Figure 12 that AS-ARBI remains effective on thwarting UNBIASED-EST with the presence of a larger k. On the other hand, Figure 13 shows that while the recall and precision inevitably drops for a larger k, the utility remains well preserved for bona fide search users, with recall above 70% and precision above 90% for almost all tested settings.

Suppression of SUM Aggregates: We also tested the effectiveness of AS-ARBI on suppressing other types of aggregates (other than COUNT(*)). In particular, we tested a SUM query which is the total length of all documents in the corpus which contain word "sports". Figure 14 depicts the results for aggregate suppression (note that utility remains unchanged as the COUNT(*) case). One can observe from the figure the effectiveness of AS-ARBI on suppressing access to SUM aggregates.

Efficiency of AS-ARBI: We tested the extra overhead incurred by AS-ARBI on query processing, and compared it with the original overhead of the search engine. Figure 15 depicts the results. One can make two observations from the figure. First, the response time sees minimum increase with the deployment of AS-ARBI. Second, the extra ratio of overhead incurred by AS-ARBI hardly changes when more queries have been processed by the search engine, indicating the scalability of AS-ARBI to a larger number of historic queries.

Evaluation of AS-SIMPLE: We also tested our basic algorithm AS-SIMPLE, with Figures 16 and 17 depicting the its effectiveness on aggregate suppression and utility, respectively. One can see that while AS-SIMPLE successfully thwarts COUNT(*) estimation by UNBIASED-EST, the utility it achieves is substantially lower than that of AS-ARBI, indicating the effectiveness of virtual query processing on improving the utility of search engine.

Problem of AS-SIMPLE: Finally, we verified the possibility of correlated-query based attack against AS-SIMPLE using two very small corpora: corpus P (1000 documents, adversarial recall 97.72%) and corpus $2P$ (2000 documents, adversarial recall 93.72%), when $k = 50$. Based on the same set of documents from which we constructed the regular adversarial query pool, we identified queries strongly correlated with query "sports" to form the pool of correlated queries. Figures 18 and 19 show the change of query COUNT when we issue 94 queries in order over P and $2P$, respectively. One can see from the figures that, when AS-SIMPLE is applied, an adversary can easily observe the decrease of query COUNT over P and thereby distinguish between P and $2P$. AS-ARBI, on the other hand, effectively suppresses such a distinction.

7. RELATED WORK

Aggregate Estimation for Search Engine: Several papers have been published in the area of search engine aggregate estimation. [11] presented an method to measure search engine coverage via random queries through public query interface. [9] significantly improved the quality of search engine aggregate estimation by using 'importance sampling'. [26] introduced 'stratified sampling' and a new document cardinality estimator, with which, efficiency of search engine aggregate estimation was greatly improved. [10] presented a new aggregate metric 'impressionrank', which imposed different weights to different web pages based on impressions given to search engine users - and thereby made estimations more meaningful from a normal user's perspective.

Privacy Protection in Data Mining: A lot of privacy protection research in data mining focused on protecting individual tuples, which is a complimentary to our work in this paper. Generally, these techniques could be classified as query auditing and value perturbation. [19, 22] are good references to query auditing and [5, 6, 13, 21, 24] are good references to value perturbation. Some research papers discussed aggregate protection, in which [7, 14, 25] presented the problem of protecting sensitive association rules in frequent pattern. [12] discussed the necessity of protecting aggregate information for hidden database and an algorithm based on dummy tuple insertion.

8. CONCLUSIONS

In this paper, we initiated a discussion on the problem of enterprise search engine aggregate suppression. We first presented algorithm AS-SIMPLE, which can effectively suppress search engine aggregates for all current estimators. Then we observed adversaries may launch attacking for AS-SIMPLE based on correlated queries. Based on this observation, we proposed AS-ARBI module, with which we can not only conquer correlated queries, but also improve utility. Comprehensive experiments have been conducted to illustrate the effectiveness our algorithms.

Our discussion is preliminary and multiple possible extensions exist. In this paper, we focused on dynamically hiding edges between documents and queries. Other possible methods could involve query auditing and dummy document insertion. Our algorithms may also be modified or extended to suppress aggregates over other online data sources - e.g., aggregate suppression over structured hidden web databases, online social networks, etc.

9. REFERENCES

[1] Open directory project `http://www.dmoz.org`.

[2] `http://code.google.com/apis/soapsearch/api_faq.html`.

[3] `http://en.wikipedia.org/wiki/AOL_search_data_scandal`.

[4] E. Adar. User 4xxxxx9: Anonymizing query logs. In *Query Logs Workshop, WWW*, 2007.

[5] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, 2000.

[6] R. Agrawal, R. Srikant, and D. Thomas. Privacy preserving olap. In *SIGMOD*, 2005.

[7] M. Atallah, E. Bertino, A. Elmagarmid, M. Ibrahim, and V. Verykios. Disclosure limitation of sensitive rules. In *KDEX*, 1999.

[8] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine's index. In *WWW*, 2006.

[9] Z. Bar-Yossef and M. Gurevich. Efficient search engine measurements. In *WWW*, 2007.

[10] Z. Bar-Yossef and M. Gurevich. Estimating the impressionrank of web pages. In *WWW*, 2009.

[11] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. *Computer Networks and ISDN Systems*, 1998.

[12] A. Dasgupta, N. Zhang, G. Das, and S. Chaudhuri. Privacy preservation of aggregates in hidden databases: why and how? In *SIGMOD*, 2009.

[13] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. *Theory of Cryptography*, 2006.

[14] A. Gkoulalas-Divanis and V. Verykios. An integer programming approach for frequent itemset hiding. In *CIKM*, 2006.

[15] Y. Hong, X. He, J. Vaidya, N. Adam, and V. Atluri. Effective anonymization of query logs. In *CIKM*, 2009.

[16] A. Hulgeri, G. Bhalotia, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword search in databases. *Bulletin of the Technical Committee on Data Engineering*, 2001.

[17] X. Jin, N. Zhang, A. Mone, and G. Das. Randomized generalization for aggregate suppression over hidden web databases. *PVLDB*, 2011.

[18] R. Jones, R. Kumar, B. Pang, and A. Tomkins. I know what you did last summer: query logs and user privacy. In *CIKM*, 2007.

[19] K. Kenthapadi, N. Mishra, and K. Nissim. Simulatable auditing. In *PODS*, 2005.

[20] R. Kumar and S. Vassilvitskii. Generalized distances between rankings. In *WWW*, 2010.

[21] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *TKDD*, 2007.

[22] S. Nabar, B. Marthi, K. Kenthapadi, N. Mishra, and R. Motwani. Towards robustness in query auditing. In *PVLDB*, 2006.

[23] B. Poblete, M. Spiliopoulou, and R. Baeza-Yates. Website privacy preservation for query log publishing. *Privacy, Security, and Trust in KDD*, 2008.

[24] L. Sweeney et al. k-anonymity: A model for protecting privacy. *IJUFKS*, 2002.

[25] V. Verykios, A. Elmagarmid, E. Bertino, Y. Saygin, and E. Dasseni. Association rule hiding. *Knowledge and Data Engineering*, 2004.

[26] M. Zhang, N. Zhang, and G. Das. Mining a search engine's corpus: efficient yet unbiased sampling and aggregate estimation. In *SIGMOD*, 2011.