

Efficient and Private Approximations of Distributed Databases Calculations

Philip Derbeko Shlomi Dolev Ehud Gudes^{*} and
Jeffrey D. Ullman^{**}

^{*}philip.derbeko@gmail.com, dolev@cs.bgu.ac.il, ehud@cs.bgu.ac.il
Ben-Gurion University of the Negev, Computer Science
Department, Beer-Sheva, Israel

^{**}ullman@gmail.com Stanford University, Stanford, CA, USA

May 23, 2016

Abstract

In recent years, an increasing amount of data is collected in different and often, not cooperative, databases. The problem of privacy-preserving, distributed calculations over separated databases and, a relative to it, issue of private data release were intensively investigated. However, despite a considerable progress, computational complexity, due to an increasing size of data, remains a limiting factor in real-world deployments, especially in case of privacy-preserving computations.

In this paper, we present a general method for trade off between performance and accuracy of distributed calculations by performing data sampling. Sampling was a topic of extensive research that recently received a boost of interest. We provide a sampling method targeted at separate, non-collaborating, vertically partitioned datasets. The method is exemplified and tested on approximation of intersection set both without and with privacy-preserving mechanism. An analysis of the bound on error as a function of the sample size is discussed and heuristic algorithm is suggested to further improve the performance. The algorithms were implemented and experimental results confirm the validity of the approach.

1 Introduction

Consider different data providers holding vertically partitioned data. Each data provider holds different information about the same set of individuals and there is a common identifier (social security number, or any other sort of ID) that allows one to cross-reference individuals across data providers. For example, the data providers contain a data of a set of individuals: gene “banks” hold

genetic information, police departments hold criminal records, financial institutions keep record of credit history, hospitals record health history of patients for future diagnosis, etc. Those data providers might be geographically spread, belong to different organizations and have their specific privacy and security requirements. For instance, a data provider of genetic information might not allow a public access to its data while allowing access for medical doctors to their patients data, or police departments that allow data access for international law enforcements, but deny access to anyone outside the departments. In such cases, it is usually impossible to gather all the data in one place, either due to the size of the data or due to privacy restrictions on data sharing. A client issuing a query to a different data providers in public cloud settings, might be required to pay for utilized CPU time and network traffic. In many cases, such client will be willing to trade performance (in other words, cost) for accuracy of the answer. Especially, if the trade off is controlled by the client, which can define the accepted error in the received answer.

In this paper we propose a method of trading off performance for accuracy using sampling. We suggest a specific way to perform sampling in vertically split datasets and calculate sample size given acceptable error. Performance improvements of the method are shown on a calculation of intersection set cardinality, together with a proposed heuristic algorithm. We also discuss a case of non-cooperative data providers and adaptation of the proposed algorithm for differential privacy calculations.

Our goal: given a set of k data providers

$$DP = \{D_1, D_2, \dots, D_k\},$$

which data records share a common identifier w and a set of predicates $P = \{p_1, p_2, \dots, p_k\}$, find the size of intersection $\cap_i p_i(D_i)$ where $p_i(D_i)$ is a set of all records in D_i that satisfy predicate p_i .

We assume that the number of records that are not present in all datasets is very small compared to the total number of records and thus, for the sake of simplicity, the data providers in DP are assumed to contain the same records with size denoted by $N = |D|$, i.e. vertical split of the data.

The size of the intersection might not be calculated precisely, but up to a given accuracy ϵ provided by the client querying the data providers. For example, the client that performs a query is interested in an integer percentage of people from the entire population having both criminal record and specific genetic mutation, i.e. the results can be rounded to a closest percentage. Intuitively, this relaxed requirement should result in more computationally and network efficient algorithms. The optimization of computation time and network traffic are important, as mentioned above, in the commonly used public-cloud deployment; the user is charged for computation time (also called CPU time) and network traffic. Thus, minimization of those parameters is a very attractive algorithm property and present a trade-off with accuracy of the intersection set size.

Related work and our contribution: Conjunctive queries over distributed databases were extensively researched (see as an example [24, 26, 12, 3, 42]).

Previous research has references both the aggregation techniques and performance issues of such queries. However, those works are based on an assumption that databases freely share the data, which does not hold in a settings of non-cooperative data providers.

There were a number of works describing privacy preserving calculations of intersection size, see [32, 18, 23, 8, 30, 37, 1]. Most of those works described a protocol that performs an exact calculation of the intersection set using Secure Multiparty Computations (MPC), first investigated by [43] and later generalized to multiparty computation. Performing MPC protocols allows efficient calculation of the intersection size, however, when used by itself, it is also possible to leak information about specific items in the datasets. For instance, the user can issue a query knowing with a single possible answer and check whether the intersection set is empty or not. Such technique allow the user to identify the presence or absence of a specific item in the datasets. A technique that can be used to hide a presence of individual items in the dataset is differential privacy ([16]).

While some of the above works ([32]) performed inexact calculations of the intersection set, the source of approximation was to preserve differential privacy of the results. For instance, [18] applied Oblivious Transfer (OT) protocol to approximate the size of the intersection of two databases, where OT has lead to the inexact result. Our approach is to utilize the relaxed requirement of providing inexact result to improve the computational complexity of queries.

Sampling as a way to cope with huge volumes of data or to increase computation performance were considered in a number of different contexts. ([19, 20, 36]) and Section 4.2 of [26] discussed sampling as a means to cope with massive datasets by creation of a histogram-based estimators. Once the estimator is created, when possible, the database performed estimation calculations on the histogram. Reservoir stream sampling algorithms ([41, 17, 39]) were developed to provide a sample of online data. The techniques fill in the sample (“reservoir”) by knocking out existing sample items from the sample with reducing probability as the sample begins to fill. ([6, 7]) initiated usage of document sampling for document similarity comparison on Internet scale. Ideas were later developed into min-wise sampling techniques ([34, 9, 11, 10]), which appear to be more suited for horizontal split datasets. In addition, a considerable corpus of work exists on concentration inequalities in scope of Machine Learning. Concentration inequalities investigate the relation between the size of the sample and its statistical similarity to the entire dataset. PAC-Bayes bounds on hypothesis error as a function of sample size were provided in [28, 29, 25].

In this paper, we consider the case of approximate calculations in distributed databases where the data is split vertically. We suggest an algorithm that takes advantage of a lack of accuracy in a distributed answer to considerably speed up the queries. We show a use-case of an algorithm on intersection set cardinality calculations both with and without privacy considerations. The algorithm adopts sampling techniques from ([19, 20]) and uses techniques similar to PAC-Bayes bounds from ([28, 29, 25]) to decide on a minimal, representative sample size. Performance improvement might be especially significant in privacy

preserving setting, where the calculations take considerable time. Specifically our contributions are:

- Suggested efficient method for approximate, distributed calculations with vertical-split datasets.
- Proposed a method of choosing sampling size given a required error level and provided simulation results of comparing the error level of different sample sizes.
- Suggested a heuristic algorithm of speeding up the intersection set size calculation based on bounds convergence and showed its adaptation for privacy preserving intersection set size calculation.

The rest of the article is structured in the following way. Section 2 defines a naïve algorithm for calculation of exact intersection set size, Section 3 relaxes the requirement to calculate the exact size of intersection set and defines theoretical bounds for *inexact* intersection. Section 4 presents results of experiments that validate the proposed sampling method. Section 5 describes a heuristic algorithm for calculation of intersection based on the bounds of intersection size. Finally, Section 6 discusses privacy issues in the described algorithms. The paper is concluded in Section 7.

2 Naïve Algorithm

The idea of the Naïve Algorithm is simple: iterate over data providers $DP = \{D_1, D_2, \dots, D_k\}$ and exchange a monotonously decreasing set of keys that satisfy all predicates up to the current iteration. The data provider then checks which of the received keys answer the corresponding predicate and returns a new list to the client. The client continues the process until all data providers were queried and the intersection set is found. In this way, the client calculates the intersection set iteratively. Denote this set by L_i where i is the iteration number. In other words, $L_1 = p_1(D_1)$ where $p_i(D_i)$ is a set of records in D_i that satisfy predicate p_i , and $L_i = L_{i-1} \cap p_i(D_i)$.

The Naïve Algorithm can also be performed in a parallel way, where the client receives a set of records that satisfy the corresponding predicate from each data provider. The client then calculates the intersection set. Comparing the sequential and parallel naïve algorithms, sequential algorithm optimizes the network load and also improves CPU time, but not a wall-clock time of the algorithm (i.e., the time between the beginning of the operation and the time the client gets the final answer). Unless stated otherwise, in the rest of the section we consider only the sequential version of the algorithm.

Despite its virtue of being simple, the naïve algorithm has a few drawbacks.

- The amount of information transferred between the client and the data providers is relatively large, as the entire set of keys in the intersection set is sent.

- The first data providers evaluate the given predicate over their entire dataset, which is expensive.
- The wall-clock time of the sequential algorithm is linear in a number of data providers and the size of the data.
- The algorithm completely exposes information both between data providers and between data providers to the client.

As mentioned above, those drawbacks are even more extreme in a current common practice, where a public cloud infrastructure is used for calculations, as in a public cloud infrastructure the client is charged for computation time (CPU time) and for network traffic. Thus, there is a monetary incentive to minimize those two values.

Both the sequential and the parallel naïve algorithms described above calculate the intersection set exactly, thus having a relatively high network traffic and CPU time demands. The next section discusses a way to reduce computation and network complexity by calculation of an estimation of the intersection size.

3 Reducing Data Size by Sampling

Relaxing the requirement for exact calculation of the intersection, this section discusses a way to calculate a smaller intersection set whose size can be extrapolated to the intersection size of the entire population. Instead of calculating the intersection of the entire datasets, perform a sample and calculate the intersection of the sample. Then, scale up the size of the sample intersection to the size of the intersection for the entire population. The results of scale up will be an estimate for the intersection size.

The requirements from the sample are clear: the sample should be as small as possible while truthfully representing the entire population (given the definition and the error of representation). Those two requirements present a trade-off between sample size and the accuracy of algorithm results. A number of different sampling techniques were developed over the years: on-line (reservoir) stream sampling algorithms ([41, 17, 39]), Histogram-based estimators ([20, 19, 36]) and min-wise sampling techniques ([6, 7, 34, 9, 11, 10]) which are more suited for horizontal split datasets. In our settings, it seems natural to choose hash function based technique, which in a sense a randomized version of histogram.

To provide sufficient improvements in performance, the size of the sample (denoted by m) should be much smaller than the average size of the datasets ($|\hat{DP}|$): $m \ll |\hat{DP}|$. Therefore, if each data provider performs its own sampling, the overlap between the samples will be very small. In other words, the probability of the data record (person in our example) appearing in the sample is low. For every data provider i

$$\forall x \in D_i : P(x \in S_i) = \frac{m}{|\hat{DP}|},$$

where S_i is a sample from the dataset of the data provider i . Leading to the probability of the item appearing in all samples being (assuming that all data providers contain the same records):

$$P(x \in \cap_{j=1}^k S_j) = \prod_{l=1}^k \frac{m_l}{|D_l|},$$

where n_l and $|D_l|$ are a sample size and a dataset size of a data provider l . Assuming, for simplicity, that the sample size and dataset size are equal for all data providers, the probability becomes:

$$P(x \in \cap_{i=1}^k S_i) = \left(\frac{m}{|\hat{D}P|} \right)^k.$$

As this probability is low, the size of the intersection will be essentially zero for most of the samples. The solution is to make all data providers create the same sample. In order to do that, we will use a sampling technique based on hashing ([19]). The technique works in the following way:

1. The client defines an accuracy threshold for the calculation.
2. Using techniques described below, the size of the sample is determined: m .
3. The number of hash buckets is $b = \left\lceil \frac{|\hat{D}P|}{m} \right\rceil$.
4. Pick a hash function H that will distribute datasets of data providers D_i into b buckets. H hashes only the ID of the records and not the entire data, as the ID is the shared information across different data providers.
5. The client sends each data provider 3 parameters: H , the number of the bucket that was chosen randomly and predicate p_i .¹ This will ensure that all data providers use the same sample.
6. Each data provider evaluates its predicate on the records in a given bucket ($p_i(D_i)$) and sends the results to the client.
7. The client performs the intersection calculation on the received results.

As client acceptable error defines sample size, which is the bucket size, it is possible to pre-calculate a number of buckets for different error value. This will eliminate the need for sequential scan of the entire database on each query. However, it will also mean that the client will get approximately the required error.

Notice that just like the Naïve algorithm sampling can be done both in parallel and sequential ways. Following the similar idea of the Naïve algorithm,

¹It is possible to always use a predefined bucket with a downside that for a given size, there is a single representative sample.

in the parallel version of sampling algorithm each data provider evaluates the predicate over the entire given bucket, where in the sequential version, the client sends the current intersection set (which is a subset of the given bucket) to the next data provider, which evaluates the given predicate only over the received subset. However, in contrast to the Naïve algorithm, where the sequential variant significantly reduces the network load and CPU time, in sampling algorithm the reduction is much smaller, as the core underlying idea of the sampling algorithm is minimization of the amount of records participating in the intersection calculation. Importantly, the accuracy of the intersection size estimation is the same in both sequential and parallel versions of the sampling algorithm.

Sampling improves both the computation time and network traffic over the Naïve algorithm by the factor of $\frac{m}{|DP|}$. Thus, one important question still remains unanswered: what should be the size of the sample given an accuracy threshold? The following sections define bounds on the sample size such that with high probability it will represent behavior similar to the entire dataset. Two different cases are considered: sample size is small compared to the dataset size (Section 3.1.1) and sample size is comparable to dataset size (Section 3.1.2). Notice that in both cases the sampling is performed in the same way as explained above. However the bound used to calculate the sample size is different for those two cases.

3.1 Selecting a Sample Size

Selection of the sample size is driven by the trade-off between the accuracy of the sample and the performance of calculations. The bigger the sample, the more accurate it is, but also the larger the computation time required to calculate the intersection. Using a bound on the error as a function of the sample size, it is possible, given a limit on an acceptable error, to choose the size of the sample set.

3.1.1 Bound for Small Sample from Large Dataset

When a sample size is very small compared to the dataset size, the sampling can be approximated by independent sampling of the same size with replacement. The approximation can be done, as the probability of any record being a part of the sample goes from $\frac{1}{N}$ for the first pick to $\frac{1}{N-m}$ for the last pick. If $N \gg m$ then $N - m \approx N$, and therefore, the probability is approximately $(\frac{1}{N})$, which is the same probability for a record to be picked when sampling with replacement. The reason to perform such approximation is due to a fact that it is simpler to provide a bound for independent samples with replacement rather than for sampling without replacement.

The size of the sample should be big enough that the sample will be a good representative of the entire dataset for the intersection calculations. How can the size be estimated?

Let us consider a sampling from the dataset D_i according to the uniform distribution. Notice that even though the sampling is done according to uniform

distribution, as described in the sampling algorithm (Section 3), the datasets themselves might be drawn from other distributions. The provided bounds still hold, as the sample “similarity” to the entire dataset depends only on its size. If a different, non-uniform sampling technique is used, the bound might be changed to use more general Bernstein inequalities ([4]).

Define X_i to be a random variable that the sampled item w_i satisfies condition L . Then, the average value of the sample, denoted by \bar{X} , should be close to the mean value, μ , of the entire dataset. Notice that in the binary case, the average is simply the number of data records that satisfy a given predicate divided by the data size. In order to bound the difference between the average value of the sample and the mean value of the dataset, we can use concentration inequalities. Namely using results by Hoeffding ([21]) if X_1, X_2, \dots, X_m are independent and $i = 1, 2, \dots, m : 0 \leq X_i \leq 1$, then:

$$Pr(\bar{X} - \mu \geq \epsilon) \leq \left(\frac{\mu}{\mu + \epsilon} \right)^{(\mu + \epsilon)} \left(\frac{1 - \mu}{1 - \mu - \epsilon} \right)^{(1 - \mu - \epsilon)^n} \leq e^{-2m\epsilon^2}. \quad (1)$$

For simplicity, for now we will consider only the right-side of the equation, i.e.

$$Pr(\bar{X} - \mu \geq \epsilon) \leq e^{-2m\epsilon^2}, \quad (2)$$

where m is the size of the sample and ϵ measures the “resemblance” of the sample to the mean of the dataset. Notice that the equation does not depend on the size of the dataset, as it is considered much larger than the sample. The bound in Equation 2 is used by the client when it issues queries to the data providers by defining both the acceptable error (ϵ) and the target probability of exceeding the acceptable error.

3.1.2 Sample Size is Comparable to a Dataset Size

In cases where sampling size is comparable to the dataset size, it is possible to develop bounds directly for sampling without replacement. In the rest of the section, we show two bounds on sampling without replacement, one is based on a reduction of “randomness” of the data and the second one is based on a direct counting technique. Those bounds can be expected to be tighter than those based on reduction to independence or bounds for sampling with replacement. The reason for this as follows. Assume that k points were sampled out of N points without replacement. The next point is to be sampled from a set of $N - k$ rather than N points, which would be the case in sampling with replacement. The successive reduction in the size of the sampled set reduces the “randomness” of the newly sampled point as compared to the independent case, and also introduces dependency between samples. Whereas, the bound provided in Equation 2 does not depend on the dataset size and does not take the reduction in population size into consideration. This intuition is at the heart of Serfling’s improved bound ([38]) which is stated next. The result holds for general bounded loss functions and is established by a careful utilization of martingale techniques combined with Chernoff’s bounding method.

Theorem 1 (Serfling) Let $C = c_1, c_2, \dots, c_N$ be a finite set of non-negative bounded real numbers, $|c_i| \leq B$. Let Z_1, Z_2, \dots, Z_m be a random variables obtaining their values by sampling C uniformly at random **without** replacement. Set $Z = (1/m) \sum_{i=1}^m Z_i$. Then,

$$Pr(Z - \mathbf{E}Z \geq \epsilon) \leq \exp \left\{ - \left(\frac{2m\epsilon^2}{B^2} \right) \left(\frac{N}{N - m + 1} \right) \right\}. \quad (3)$$

Similar bounds hold for $Pr(\mathbf{E}Z - Z \geq \epsilon)$.

Compared to the bound in Equation 2, the above bound is always tighter when $N/(N - m + 1) > 1$, i.e. when $m > 1$.

In our case c_i 's are binary variables and the bound could be improved further by using a proof based on a counting argument ([13]).

3.1.3 Using Bounds to Calculate Sample Size

Figure 1 presents a single example of comparison between the above bounds (2 and 3). As expected, the Serfling bound is tighter than the Hoeffding bound and thus, using this bound for calculation of sample size will result in a smaller sample set.

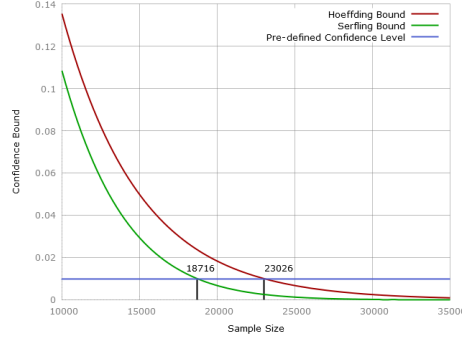


Figure 1: A comparison of Hoeffding and Serfling bounds, where $\epsilon = 0.01$, $N = 100000$. The error value is preset and then the sample size that fits this value is picked. In tighter bounds, the sample size will be smaller.

Now we can briefly describe the process of using those bounds. We assume that the client attempts to minimize the size of the sampling sets, as this also minimizes the network load and CPU time, both of which are chargeable in public cloud environments. The client defines a value of an acceptable error for the combined distributed query (E) and a “confidence” of the error.² The bounds defined in (2) or (3) provide a “confidence” that the error will be smaller than ϵ given a sample size. Now the client uses the bound from (2) or (3) to

²In a very similar way to PAC bounds ([40]) in Machine Learning.

determine the minimum sampling size that allow the required error with “high enough” confidence. Using a predefined confidence and a chosen bound, the client can find the minimum value of sample size that will provide the required error with the required confidence. For example, consider Figure 1, where for dataset size of 100,000 the client choose $\epsilon = 0.01$ and confidence of 0.01. Thus, using the simplified Hoeffding bound (2) the minimal sample size is 23,026 and using the Serfling bound (3) the minimal sample size is 18716.³

After the intersection of sample sets is calculated, there is a need to estimate the size of the intersection of the entire datasets. As mentioned above, the mean of the binary random variable is also the number of records that belong to a set defined by the predicate divided by the size of the sample. Thus, the ratio of the records in a sample that satisfy a given predicate is close to the ratio of the records that satisfy the predicate in the entire dataset of a given data provider. This leads to the conclusion that the ratio of the sample intersection set size to the sample size should be the same as the ratio of the intersection set size to the size of entire dataset. Let \hat{S}_i be a sample of D_i of size m , then

$$|\cap_{i=1}^k D_i| = |\cap_{i=1}^k \hat{S}_i| \frac{|\hat{DP}|}{m}. \quad (4)$$

Notice that even though the absolute error in the intersection estimation of the entire dataset is proportionally larger than the error in estimation of a sample, the relative error will remain the same. The reason for the error to remain the same in a process of intersection calculation is due to the sampling method. Even though the sample from each data provider has the same (potential) relative error, the error is unique for the sample and thus, it does not accumulates when the intersection is calculated.

As ϵ in the above bounds (Equation 2 and 3) is a bound on difference in estimation, it is a relative error. Therefore, if the client defines an acceptable error (E) as an absolute error, it can be easily translated: $\epsilon = \frac{E}{D}$.

4 Experiments

We have performed a number of experiments to show the utility and error resulted from sampling. The experiments were performed on simulated data and validated on the Adult dataset ([27]). The described sampling technique is targeted at large datasets with many records, such that calculation of predicates over the entire dataset is wasteful. For such datasets, it is much more practical to test the algorithm on simulated data, which can be generated on any required scale.

³This example also shows the advantage of tighter bounds, which in this case is 23% of a sample size.

4.1 Experiments on Simulated Data

For experiments on simulated data, a number of datasets were generated with random values of the predicates and then the intersection size was calculated according to each algorithm. The methodology works as follows.

- Generate a dataset of a given size. All data providers assumed to share the same set of records with possibly a different data per record.
- For each data provider generate a set of predicate values randomly, such that the frequency of “true” values is as defined.
- Calculate the intersection iteratively, i.e. by addition of a single datasets at a time. This is a both a simpler way of implementation (as opposite to parallel calculation) and also allows observation of convergence rate of the intersection size.

All experiments were averaged over 10 runs, standard deviation was calculated and drawn on the resulting graphs.

Figure 2 shows the intersection size as calculated by the sequential naïve algorithm and estimation by sampling of various sizes. The graph provides a high-level, visual practical validity of the approach. Even a relatively small sample sizes (1% of the dataset size) estimate intersection size close to the real value.

In order to focus on the error caused by sampling and show the difference in sample size more clearly, we have performed a number of experiments showing relative error between estimation and the real intersection size. The Naïve algorithm calculated was taken as a baseline for error calculations. The Sampling algorithm was executed with a number of sample sizes, each one showing the relative error from the Naïve algorithm results. Figures 3, 4, 6, and 5 show the relative error of the sampling algorithm with various sample sizes. The Y axis shows the error of the sampling, i.e. $\frac{\text{sampled estimation} - \text{intersection size}}{(\text{intersection size})}$,⁴ while the X axis is the number of datasets in the intersection. The graphs compare datasets sizes of 100,000 (Figures 3 and 6) and 1,000,000 (Figures 4 and 5) records with predicate satisfaction frequency of 0.7 and 0.5, i.e. in each data provider, 70% or 50% of the records satisfy the corresponding predicate. As mentioned above, each graph is an average of 10 different runs and standard deviation is depicted by error bars on the graph.

The graphs show that, as expected, larger samples result both in smaller error and smaller standard deviation. However, it also can be seen, that the error quickly converges as a function of sample size. In some cases, even for sample of 1% from entire dataset, it is possible to achieve reasonable error. Overall, sampling 1/10 or 1/5 of the dataset resulted in errors of approximately 0.01 from the intersection size, which is an order of a single percent. Thus,

⁴The error is relative but not normalized. It is possible to multiple the error by 100 to translate it into percentages from the intersection size. Notice that as the intersection might be relatively small, the error in percentage might be large. Comparing it to the dataset size will result in a much smaller error values.

queries of a type: “What is the approximate percentage of people ... ?” can be answered using only tenth of a data.

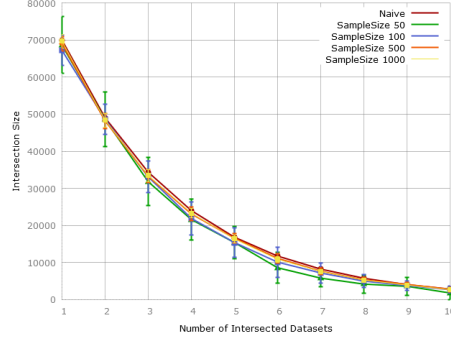


Figure 2: A comparison of the Naïve and the Sampling algorithms. Dataset size was set to $N = 100,000$, number of datasets is $k = 10$, ratio of predicate satisfaction in each dataset is 0.7. The Y axis shows the size the intersection estimation. The X axis show the number of the data sets that participated in the intersection. Error bars show standard deviation over 10 different executions.

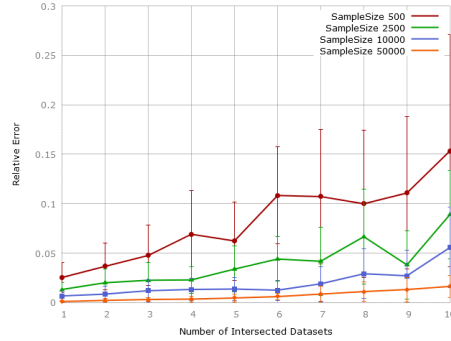


Figure 3: A comparison of errors of different sample sizes. Dataset size was set to $N = 100,000$, number of datasets is $k = 10$, ratio of predicate satisfaction in each dataset is 0.7. Error bars show standard deviation over 10 different executions.

4.2 Experiment on a Real Dataset

In addition to experiments on simulated datasets, we have tested the methodology on Adult dataset from UCI Machine Learning Repository ([27]). The dataset contains data from census bureau, where each record has data about different person. This fits a description of the setup we have described. The dataset contains 32,562 instances. As a test case we have used an intersection of

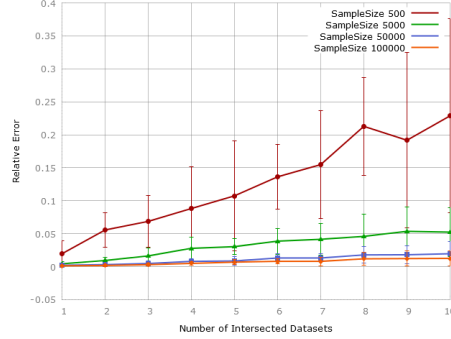


Figure 4: A comparison of errors of different sample sizes. Dataset size was set to $N = 1,000,000$, number of datasets is $k = 10$, ratio of predicate satisfaction in each dataset is 0.7. Error bars show standard deviation over 10 different executions.

the following predicates: age ≥ 30 , marital status: Never-married, Sex: Female or Male in two different tests and income $> 50K$. The exact intersection set size is 252 for males and 139 for females. Since the dataset was not used for classification task, it allowed us to use income as one of the predicates. In addition, the intersection was of 4 predicates, as addition of more predicates resulted in a small or empty intersection set due to a limited dataset size.

Samples of different sizes were drawn with sample size ratio going from 0.1 to 0.5. The intersection set size was calculated from the drawn sample. Figure 7 depicts the results of the testing. While the accuracy of the estimation does not necessary improves by taking larger samples, the standard deviation becomes smaller. The accuracy improvement is most probably caused by the small size of the intersection, while improvement in standard deviation fits the results shown in simulated datasets. Overall, the accuracy of the intersection set size calculation verify the validity of the approach.

5 Heuristic Algorithm for Bounded Estimation

Previous sections described exact and approximate ways of calculating the size of the intersection. This section presents a heuristic algorithm that attempts to optimize the calculation of the intersection by not performing the calculations for all datasets. The simplest case of heuristic is when the intersection is calculated iteratively and at some point the intersection set is empty. Clearly, the calculation can be stopped at this stage. Following is a heuristic algorithm for the intersection calculation that attempts to stop at the earliest possible point.

The size of the intersection depends on the sizes of the sets from each data provider that answer the corresponding predicate and on the correlation between those sizes. Below we suggest a heuristic algorithm for estimation of the intersection set size. The algorithm starts with an accuracy parameter, the ratio

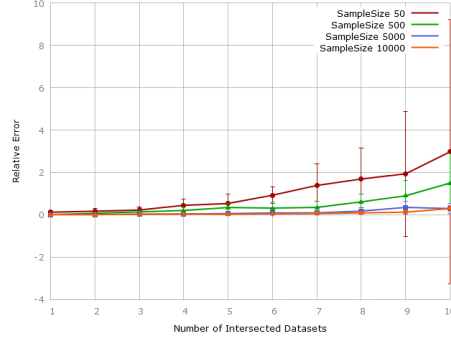


Figure 5: A comparison of errors of different sample sizes. Dataset size was set to $N = 100,000$, number of datasets is $k = 10$, ratio of predicate satisfaction in each dataset is 0.5. Error bars show standard deviation over 10 different executions.

of records that satisfy the relevant predicate in each data set, and the lower and the upper bounds on the intersection size. The algorithm iteratively tightens the bounds until the difference between the bounds is smaller or equal to the accuracy parameter. Then the algorithm stops and returns the middle value of the range between the lower and the upper bound.

5.1 Upper and Lower Bounds on the Intersection Size

Let p_i be the predicate that is associated with a data provider D_i . Then, $p_i(D_i)$ is the set of members in D_i that satisfy p_i . Also, \hat{p}_i will denote the fraction of the members that satisfy p_i in D_i , i.e. $\hat{p}_i = \frac{|p_i(D_i)|}{|D_i|}$. Now, let us define the bounds on the size of the intersection set J for the sake of the algorithm iterations. For simplicity, we assume that data provider datasets D_1, D_2, \dots, D_k contain *exactly* the same records. The bounds will hold when the difference in members between datasets is small.

As the intersection size is at most the size of the smallest predicate set, the size of the intersection is bounded from above by the following bound:

$$|J| \leq \min_i |p_i(D_i)|. \quad (5)$$

The lower bound on the intersection of two sets D_1 and D_2 is ([2])

$$|D_1 \cap D_2| \leq \begin{cases} (\hat{p}_1 + \hat{p}_2 - 1) \frac{|D_1| + |D_2|}{2}, & \text{if } \hat{p}_1 + \hat{p}_2 \geq 1. \\ 0, & \text{otherwise.} \end{cases}$$

In case of 3 sets, the lower bound becomes:

$$(\hat{p}_1 + \hat{p}_2 - 1) + \hat{p}_3 - 1,$$

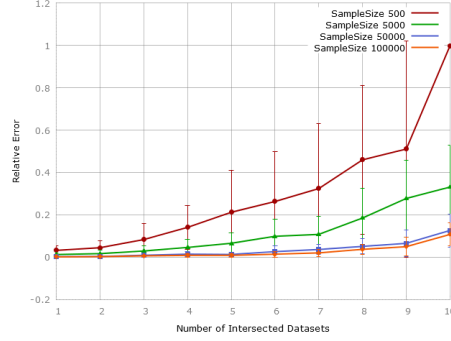


Figure 6: A comparison of errors of different sample sizes. Dataset size was set to $N = 1,000,000$, number of datasets is $k = 10$, ratio of predicate satisfaction in each dataset is 0.5. Error bars show standard deviation over 10 different executions.

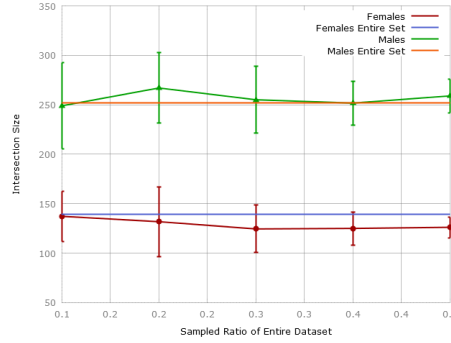


Figure 7: Comparison of intersection set size to the estimated size for different sample ratio. The experiments were performed 10 times with different samples. Average value is shown on the graph together with standard deviation results.

which in general is

$$|\cap_{i=1}^k D_i| \geq \max \left\{ \left(\sum_{i=1}^k \hat{p}_i - (k-1) \right) |D|, 0 \right\}, \quad (6)$$

where $|D|$ is the average size of the datasets (as they contain the same set of records).

5.2 The Heuristic Algorithm

As described above, the iterative step of the algorithm is to tighten the bounds on the intersection size. The algorithm will then stop once the difference between bounds is smaller than the required accuracy. As the upper bound (Equation 5)

is the minimal predicate set, an iteration step will attempt to make the minimum value smaller by calculating the intersection between the two smallest predicate sets. The intersection of two sets will be smaller or of the same size as minimal predicate set and will replace those two sets in the bound.

Calculating the intersection of two minimal predicate sets is also a good technique for increasing the lower bound (Equation 6) as well.⁵ Notice that each iteration step of the algorithm also decreases the value of k and thus removes the subtractive member of the lower bound.

The algorithm steps are as follows.

1. Define a required accuracy threshold: $\delta \geq 0$. As δ is a relative error, the iterations continue until an absolute error is smaller than $\delta|\hat{DP}|$. When the iterations stop, the middle of the bounds range is returned, therefore, the iterations stop when the distance between the bounds is less or equal to $\frac{(\delta|\hat{DP}|)}{2}$.
2. Calculate $p_i(D_i)$ for every data provider i .
3. Calculate the lower (B_l) and the upper bounds (B_u), given p_i .
4. While $B_u - B_l > \frac{(\delta|\hat{DP}|)}{2}$ and the number of sets is larger than 1:
 - (a) Let $p_j = \min_i p_i(D_i)$ and $p_\ell = \min_{i \neq j} p_i(D_i)$. In other words, pick two minimal predicate sets: j and ℓ .
 - (b) Calculate the intersection between those two sets.
 - (c) Replace $p_j(D_j)$ and $p_\ell(D_\ell)$ with the new predicate set: $p_{jl}(D_{jl}) = p_j(D_j) \cap p_\ell(D_\ell)$.
 - (d) Update new values of B_u and B_l .
5. If only one set remains p' ($p' = p_{1,2,\dots,k}$), then the size of its predicate set, $p'(D') = p'(D_{1,2,\dots,l})$, is the size of the intersection set.
6. If there is more than one set and $B_u - B_l < \delta|DP|$, then the size of the intersection set to the middle value of $[B_l, B_u]$ range: $\frac{B_u+B_l}{2}$.

The algorithm clearly converges, as the iterations stop when the intersection size is calculated exactly. At this point the lower bound is equal to the upper bound and thus, the accuracy requirement will be satisfied. Notice that if step 4b results in an empty set, the algorithms also stops, as the lower and the upper bounds will be equal and zero.

Example: As an example of the algorithm execution, assume $k = 4$, equal size of all data sets N and the required accuracy of 10%. Let the respective ratios be $p_1 = 0.8, p_2 = 0.9, p_3 = 0.5$ and $p_4 = 0.4$. In this case, the upper bound on the intersection set size will be $Np_4 = 0.4N$, whereas the lower bound will be 0, as the sum of all ratios is 2.6 less than $k - 1 = 3$.

⁵This can be seen especially in cases where there are two or more predicate sets that are smaller than 0.5. In this case the lower bound of their intersection is 0.

The first iteration of the algorithm will be to find the intersection of D_4 and D_3 . Let us assume that their intersection results in $p_{3,4} = 0.31$. Now, the upper bound of the intersection becomes $0.31N$, whereas the lower bound increases to $0.8 + 0.9 + 0.31 - (3 - 1) = 0.1N$. The iterations continue until the bounds converge to within $0.1N$.

In general, the Heuristic Algorithm will work fine in cases where the ratio of records that satisfy the predicate is high across most data providers, but the intersection size might decrease relatively fast. See Figure 8 for example of convergence of the algorithm bounds on simulated data.

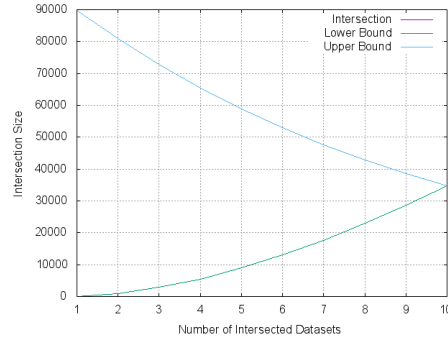


Figure 8: Convergence of the Heuristic Algorithm bounds. Dataset size was set to $N = 100,000$, number of datasets is $k = 10$, ratio of predicate satisfaction in each dataset is 0.9 (the high ratio is required for the lower bound to be larger than zero for $k = 10$). Notice that in simulated case, the intersection size is co-located with the upper bound of the intersection.

5.3 The Heuristic Algorithm Combined with Sampling

Clearly, both the computation time and network traffic depend heavily on the calculation of the intersection between the two sets in step 4b of the Heuristic Algorithm. To improve the performance of this step it is possible to use the sampling technique from Section 3. Instead of performing intersection between two sets i and j , the algorithm will calculate the intersection of two samples of those sets. However, sampling introduces an error into calculation of the intersection, which has to be related to the accuracy threshold δ of the heuristic algorithm.

Assuming that the error in sampling distributes proportionally among datasets and noting that each iteration “eliminates” one dataset, we can define the following changes to the algorithm. Every sample introduces error of $\delta' = \frac{\delta}{k-1}$. To accommodate this error, the algorithm will decrease the required accuracy threshold by this amount on each iteration. This will ensure that required by user accuracy threshold will be honored. Two steps are changed in the algorithm:

1. Step 4 then becomes:
While $B_u - B_l > (\delta - \frac{\delta}{k-1}i) \frac{|DP|}{2}$, where i is the iteration number, and the number of sets is larger than 1.
2. Step 4b becomes:
Calculate the intersection between those two sets: j and l by sampling with error threshold $\frac{\delta}{k-1}$.

The rest of the algorithm remains the same, including required accuracy threshold value.

6 Privacy of Intersection Calculations

In the described setting there are 2 different types of actors: data providers and client. There are different privacy concerns between those actors. One privacy concern is preserving the privacy of data providers. The client querying the data providers should not learn the identity of the records that are part of the intersection set or the number of records that satisfy a specific predicate in any single data provider. Notice that the client should be limited to a reasonable number of queries (sub-linear in a data size), as it is not possible to answer a large number of arbitrary queries while preserving privacy ([14, 5]). Another privacy concern is in keeping a privacy of records in a specific data provider from other data providers. For instance, a data provider might be interested in hiding the presence of specific record from other data providers.

First, it is imperative to note that the sampling (see Section 3) provides a very naïve form of privacy. As every data set record has a $1 - \frac{m}{N}$ chance of not being included in sample, the presence of the specific record in the data set is not immediately observable. Yet, the fact that the client gets record identifiers of each data provider is undesirable. Data providers can encrypt the sampling indices to hide the identity of the records that are sent to the client. Even though an additional encryption will obscure the identity of those items from the client, it still will expose the presence of individual records to other data providers. In addition, using an additional knowledge, the client can easily learn the presence of a specific record in the data providers.

6.1 Secure Multi-Party Computation (MPC)

One way to preserve privacy is to use secure multiparty computations to calculate the intersection set ([32, 18, 23, 8, 30, 37]) A simple method that uses commutative cryptography (another approach is to use secret sharing) is described in [22]. Each data provider encrypts record identifiers using its own commutative key and passes the key to other data providers. Once all data providers have encrypted all keys, it is possible to find the intersection using encrypted identifiers utilizing commutativity of the encryption and calculate its size.

There are a few downsides to using secure multi party computations. First, performance is heavily impacted by performing secure computations using these methods. It is possible to alleviate the performance issues by using the sampling to reduce a number of records in the calculated intersection, as described in Section 3. Thus, the idea is to perform the sampling algorithm and then calculate the intersection in a secure way using the MPC computations between the data providers. This method however, will still require calculation of the sampled intersection exactly, without the ability to halt when required accuracy was reached, as it is possible in heuristic algorithm. The second drawback is specific for the described use-case where the data is kept in different organizations. Secure multi party computations require direct communication between data providers, which in some cases is very challenging in a real-world deployments due to both security and technological reasons. While it is still possible for the client to act as an intermediary between different data providers, such setup also doubles the network traffic. A different approach is only to allow communication between client and data provider with adopted algorithms.

However, the most significant drawback of MPC is that the exact intersection size is calculated. This might allow the client to use additional information to infer a presence of a specific record in the dataset.

6.2 Differential Privacy

The current de-facto privacy standard is differential privacy ([16]). Informally, the idea of differential privacy is to protect the privacy of individual records in the dataset without any assumption on the additional knowledge. Differential privacy is preserved if it is practically impossible to identify whether the record is present or absent in the dataset from the result of database queries. The most common methods of ensuring differential privacy rely either on addition of a carefully chosen amount of noise to the result or by using an exponential mechanism that chooses the output according to some specific probability distribution.

The protocols that are based on secure multiparty computations perform the exact calculation of the set, which is impossible in differential-privacy settings. This led to several works describing differential-privacy-preserving calculations of the intersection size, see [32, 30]. Even though those algorithms preserve differential privacy, the requirement to provide an approximate answer in our case allows our scheme to optimize the algorithm for performance by performing secure computations on as little number of the records as possible due to sampling.

The most common method of ensuring differential privacy is an addition of a specifically crafted random noise to the released data ([16, 15]). Moreover, there are two different approaches for the private data release: interactive ([14]) and non-interactive ([31]). Interactive data release is when the client sends data query to a data provider. The data provider then releases the data to the client while ensuring differential privacy of records in its database. The data provider might decide not to answer a specific queries or to stop answering queries from

a given client, as this might impact the privacy. In non-interactive data release, the noise is applied only once and then the data is released to the client. The client can perform any number and any type of queries on the data. Intuitively, non-interactive release requires to add more noise to the released data, as it has to be ready for any query, while interactive release can adopt added noise to the results of each query ([16]).

6.2.1 Where To Add Noise in Heuristic Algorithm

In the described above sampling algorithm (Section 3), there are two intuitive locations to add random noise. First, it is possible to add noise in hashing function that assigns records to buckets. Such noise will preserve privacy of individual records, as it will be impossible to distinguish whether a specific record is within the bucket or not present at all in the dataset. A second location to add the noise is during a query processing. Since the data provider exposes only the number of records that satisfy a given predicate, the noise can be added to the output after the predicate was evaluated or to the predicate itself [[Ehud: this is not clear, the noise should be fake IDs]].

The difference between those two locations is exactly the difference between interactive and non-interactive data release. Adding noise to the hashing function is a one-time operation and has no relationship to the result of the specific predicate. Thus, this noise addition can be seen as non-interactive data release⁶. On the other hand, if the noise is added after the predicate is evaluated over records in a bucket, then the noise value can be adapted to the results of predicate evaluation. Thus, the amount of noise can be directly related to the results of a specific query, like in interactive data release settings.

Due to the above reasoning, the next section describes an addition of a noise to the predicate function results.

6.2.2 Adding Noise to an Output of Predicate Evaluation

Differentially private calculation of a counting function (referred in the paper as “noisy sum”), i.e. counting a number of records that satisfy a given predicate, was considered previously in ([33, 5]). ([16]) showed that adding a Laplacian noise $Y \approx \text{Lap}(1/\epsilon)$ to the sum query output: $\sum_i x_i + Y$, ensures ϵ -differential private function. Notice that the sensitivity of counting function is $S(f) = 1$ and thus, the distribution standard deviation is $S(f)/\epsilon = 1/\epsilon$.

The fast algorithm for privacy-preserving intersection calculation is described below. Heuristic algorithm from Section 5 is used as a basis for privacy preserving algorithm.

⁶It is possible to execute hash function to divide records into buckets on every query. However, even in this case, there is no relationship between hash functions and the number of records that answer a specific predicate. Therefore, the added noise is agnostic for the processed query and there is also a performance penalty in constant calculation of hash function over the entire database.

1. Use a hash function H to divide data into buckets. In case only a single bucket is used for all queries, only the agreed bucket is kept.
2. When a predicate is received from the client, evaluate the predicate over the chosen bucket.
3. Add random noise according to Laplace distribution to the size of the predicate set. The noisy set-size is shared with the client.
4. The client gathers results from all data providers and calculate upper and lower bounds.
5. The client continues to execute heuristic algorithm and picks two data providers, j and k , to perform intersection according to the algorithm step 4b.
6. Perform privacy-preserving, secure intersection between two chosen providers using the below algorithm.
7. Update the bounds and continue until the bounds are close enough.

6.2.3 Differentially Private Set Intersection

The algorithm used for privacy-preserving, secure intersection set calculation is based on a work of ([32]), which allows a differentially private calculation of an intersection over multiple databases. The algorithm provides computational differential privacy, as it relies on homomorphic encryption and makes some assumptions on computational hardness. This section describes how the algorithm from ([32]) can be used in our heuristic algorithm from Section 5 for inexact computations.

The algorithm describes a basic operation *private set-intersection cardinality (PSI-CA)* and then adds noise to ensure differential privacy, thus resulting in BN-PSI-CA. PSI-CA operation is based on usage of homomorphic cryptosystem that allows addition and multiplication by constant (for instance, Paillier's cryptosystem ([35])). When two data providers, i and j , attempt to calculate intersection, i defines a polynomial whose roots are the members of its set, The polynomial coefficients are then encrypted and transferred to j data provider, which calculates $Enc(rP(x_i) + 0^+)$. In other words, it evaluates the polynomial on its set members, multiplies by a constant number r and adds a special string of zeros 0^+ . When transferred back to the first data provider (i), it calculates the number of zero string. In order to add differential privacy, a random number of dummy values are added to the transferred set by both parties (for details see ([32]) rounding the set size to the bucket size in our case. In the end, data provider i learns the noised cardinality and data provider j knows the amount of noise it added to the intersection set size.

Adopting this algorithm for our case, the client performs the following actions. Assume that two data providers are chosen for intersection in step 5: j and k .

1. Perform BN-PSI-CA algorithm to calculate a noisy intersection set. Assume that without loss of generality, j will hold the size of a noisy set, where k will hold the amount of added noise.
2. j sends the size of a noisy set to the client.

In the following iterations of the heuristic algorithm, BN-PSI-CA might be invoked to find intersection size of j , k and r , and more additional data providers. As due to privacy issues, no single data provider holds the intersection dataset, it is necessary to perform intersection size calculations over again for any additional data provider.

Notice that in cases where data providers do not communicate directly, the client acts as an intermediary for the protocol. In the end, the client has a value of intersection size which is still preserving differential privacy. The client also keeps record that k holds the amount of noise of the last intersection. As described in ([32]), the noise added in one intersection, might be removed if there is a need to calculate another intersection of j and k with additional data provider.

([32]) performance results show approximately linear dependency of an algorithm run-time in the number of records in the participating datasets. Using results from Section 4, with a reasonable error, it is possible to reduce the number of records by a factor of 5-10 (according to acceptable error), thus tens of minutes to single minutes. Such a reduction might make a difference between practical and impractical system.

The algorithm accelerate approximate intersection calculation by:

- Computationally intensive calculations of intersection sizes are performed on samples. Those calculations are demanding computationally, thus, any reduce in the size of datasets is important. [[Ehud: we should try to find a third-year student to implement this]]
- The intersection is not calculate exactly, but rather the algorithm stops when the difference between bounds is close enough.

7 Concluding Remarks

A steady and rapid increase in the amount of data has resulted in an abundance of various data providers. More data is kept for longer and in more places. In parallel, the awareness of data privacy and security is also on a rise both from government regulation and personal perspective. This trend requires new algorithms and protocols for dealing with distributed data. Those new methods should be both efficient and privacy preserving.

The main practical downside of the current privacy preserving computations is the run-time complexity. Both MPC, Oblivious-Transfer and Private-Information-Retrieval methods that are used to perform distributed calculations in privacy-preserving manner significantly increase the running time of

the query. While some advances in improving performance were made, the running time still remains a limiting factor. Our proposed method uses the ability to provide an approximate answer to the supplied query and thus to reduce considerably the dataset that is used for computations. This decouples the use of sampling technique for reducing the size of the dataset used for calculations from the protocols designed to preserve privacy of individual records or data providers. As such, most of the above protocols can be used in conjunction with sampling techniques to perform approximate calculations with improved performance.

As showed in this work, when acceptable, calculating an approximate answer can significantly improve computation time. In those cases we have suggested a way to use this possibility for approximate answering by using sampling of the datasets. The sampling leads to a dramatic reduction in the size of data required for calculation: 5 – 10% of the data as observed in simulations. Such reduction of the data size is much more significant when calculations are done in a secure and privacy-preserving way.

References

- [1] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 86–97, New York, NY, USA, 2003. ACM.
- [2] A. Aho and J. D. Ullman. *Foundations of Computer Science*. 1994.
- [3] T. J. Ameloot, G. Geck, B. Ketsman, F. Neven, and T. Schwentick. Parallel-correctness and transferability for conjunctive queries. *CoRR*, abs/1412.4030, 2014.
- [4] S. Bernstein. *Theory of Probability (Russian)*. 1927.
- [5] A. Blum, C. Dwork, F. Mcsherry, and K. Nissim. Practical privacy: The sulq framework. In *In PODS*, pages 128–138. ACM, 2005.
- [6] A. Z. Broder. On the resemblance and containment of documents. In *In Compression and Complexity of Sequences (SEQUENCES97)*, pages 21–29. IEEE Computer Society, 1997.
- [7] A. Z. Broder. Filtering near-duplicate documents. In *Proc. FUN 98*, 1998.
- [8] C. Clifton and J. Vaidya. Secure set intersection cardinality with application to association rule mining. In *Accepted for Publication in the Journal of Computer Security, IOS*. Press, 2004.
- [9] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, pages 441–453, 1997.

- [10] E. Cohen and H. Kaplan. What you can do with coordinated samples. *CoRR*, abs/1206.5637, 2012.
- [11] E. Cohen, H. Kaplan, and S. Sen. Coordinated weighted sampling for estimating aggregates over multiple weight assignments, 906.
- [12] J. Dean and et al. Mapreduce: Simplified data processing on large clusters, 2004.
- [13] P. Derbeko, R. El-Yaniv, and R. Meir. Explicit learning curves for transduction and application to clustering and compression algorithms. *J. Artif. Intell. Res. (JAIR)*, 22:117–142, 2004.
- [14] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *In PODS*, pages 202–210. ACM Press, 2003.
- [15] C. Dwork. Differential privacy. In *ICALP*, pages 1–12, 2006.
- [16] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, pages 265–284, 2006.
- [17] P. S. Efraimidis and P. G. Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97:181–185, 2006.
- [18] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. pages 1–19. Springer-Verlag, 2004.
- [19] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *In Proceedings of the 27th International Conference on Very Large Data Bases*, pages 541–550.
- [20] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. pages 466–475, 1997.
- [21] W. Hoeffding. Probability inequalities for sums of bounded random variables, 1962.
- [22] W. Jiang and C. Clifton. A secure distributed framework for achieving k-anonymity. *The VLDB Journal*, 15(4):316–333, 2006.
- [23] L. Kissner and D. Song. Privacy-preserving set operations. In *in Advances in Cryptology - CRYPTO 2005, LNCS*, pages 241–257. Springer, 2005.
- [24] P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. Technical report, 2011.
- [25] J. Langford and J. Shawe-taylor. Pac-bayes & margins. In *Advances in Neural Information Processing Systems 15*, pages 439–446. MIT Press, 2002.

- [26] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.
- [27] M. Lichman. UCI machine learning repository, 2013.
- [28] D. McAllester. Simplified pac-bayesian margin bounds. In *Learning Theory and Kernel Machines*, volume 2777 of *Lecture Notes in Computer Science*, pages 203–215. Springer Berlin Heidelberg, 2003.
- [29] D. A. McAllester. Pac-bayesian model averaging. In *In Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 164–170. ACM Press, 1999.
- [30] N. Mohammed, D. Alhadidi, B. Fung, and M. Debbabi. Secure two-party differentially private data release for vertically partitioned data. *Dependable and Secure Computing, IEEE Transactions on*, 11(1):59–71, Jan 2014.
- [31] N. Mohammed, R. Chen, B. C. Fung, and P. S. Yu. Differentially private data release for data mining. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’11, pages 493–501, New York, NY, USA, 2011. ACM.
- [32] A. Narayan and A. Haeberlen. Djoin: differentially private join queries over distributed databases. In *In Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation*, 2012.
- [33] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC ’07, pages 75–84, New York, NY, USA, 2007. ACM.
- [34] R. Pagh, M. Stöckel, and D. P. Woodruff. Is min-wise hashing optimal for summarizing set intersection? In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS’14, Snowbird, UT, USA, June 22-27, 2014*, pages 109–120, 2014.
- [35] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology, EuroCrypt*, pages 223–238, 1999.
- [36] V. Poosala. Selectivity estimation without the attribute value independence assumption. pages 486–495, 1997.
- [37] M. Sepehri, S. Cimato, and E. Damiani. Privacy-preserving query processing by multi-party computation. *The Computer Journal*, 2014.
- [38] R. J. Serfling. Probability inequalities for the sum in sampling without replacement. *Ann. Statist.*, 2(1):39–48, 01 1974.

- [39] A. B. Sunter. List sequential sampling with equal or unequal probabilities without replacement. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 26(3):261–268, 1977.
- [40] L. G. Valiant. A theory of the learnable, 1984.
- [41] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, Mar. 1985.
- [42] D. Woodruff and Q. Zhang. When distributed computation is communication expensive. In Y. Afek, editor, *Distributed Computing*, volume 8205 of *Lecture Notes in Computer Science*, pages 16–30. Springer Berlin Heidelberg, 2013.
- [43] A. C.-C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167, Oct 1986.