Stratified-Sampling over Social Networks Using MapReduce

Roy Levin
IBM Haifa Research Labs
and Technion – Israel Institute of Technology
royl@il.ibm.com

Yaron Kanza
Jacobs Technion-Cornell Innovation Institute,
Cornell Tech
and Technion – Israel Institute of Technology
kanza@cs.technion.ac.il

ABSTRACT

Sampling is being used in statistical surveys to select a subset of individuals from some population, to estimate properties of the population. In *stratified sampling*, the surveyed population is partitioned into homogeneous subgroups and individuals are selected within the subgroups, to reduce the sample size. In this paper we consider sampling of large-scale, distributed online social networks, and we show how to deal with cases where several surveys are conducted in parallel—in some surveys it may be desired to share individuals to reduce costs, while in other surveys, sharing should be minimized, e.g., to prevent survey fatigue. A *multi-survey stratified sampling* is the task of choosing the individuals for several surveys, in parallel, according to sharing constraints, without a bias. In this paper, we present a scalable distributed algorithm, designed for the MapReduce framework, for answering stratified-sampling queries over a population of a social network. We also present an algorithm to effectively answer multi-survey stratified sampling, and we show how to implement it using MapReduce. An experimental evaluation illustrates the efficiency of our algorithms and their effectiveness for multi-survey stratified sampling.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*query processing, distributed databases*

General Terms

Algorithms, Experimentation

Keywords

Stratified sampling; social networks; MapReduce

1. INTRODUCTION

Online social networks are a major sources of social data, and thus, they are being used for statistical studies and market research. According to the 2013 Q1 financial report of Facebook, their network comprises 1.1 billion users, out of which 665 million are active daily, generating massive amounts of data (e.g., producing 4.5 billion "likes" daily). In recent years, the ability to effectively process such huge amounts of data has become a key factor in driving business decisions. Sampling is a common and effective way to do so [13], thus, there is a growing interest in sampling of online social networks [6, 10, 11]). In particular, sampling is used for exploratory data analysis in Facebook [7].

One of the most commonly-used techniques for sampling is *stratified sampling*, where the examined population is divided into distinct subgroups, called *strata*, and a predefined number (or percentage) of individuals is selected from each stratum [1, 5]. An answer to a sampling query is a set of individuals selected from the surveyed *population*, such that the number of individuals from each stratum is as specified in the query. When the individuals of each stratum are chosen randomly, the answer is a *representative sample*, because it correctly represents the population according to the partition to strata. To see why a stratified sample can be more effective than a simple random sample, consider the following scenario.

EXAMPLE 1. *A market research company needs to acquire information from Facebook to study how activities in an online social network correspond with the buying habits of individuals. To reduce costs, e.g., the cost of testing whether selected users are genuine, the sample should be as small as possible, yet it should also be representative of the entire population in terms of the individuals' online activities, e.g., the content they upload, their "likes", etc.*

In Example 1, a straightforward approach is to draw a simple random sample of the population. However, the more diverse the population is, with respect to the desired property, the larger the required sample size should be [8]. Stratified sampling allows to decrease the size of the survey. For instance, suppose individuals above the age of 70 have a unique online behavior, but very few such individuals are active in the network. Making sure they will be properly represented in the sample requires a large simple random sample. In a stratified sampling, they will be surveyed as a separate group and will not affect the sizes of the other groups. Therefore, stratified sampling can significantly reduce the sample size without reducing the representativeness of the sample. Another example in which producing a stratified sample of the population can be helpful is when creating a dataset for training a machine learning algorithm

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2588555.2588577.

while requiring that some specific subsets of the population will be properly represented in the dataset.

In this paper we deal with two cases: *(a)* answering a single stratified sampling query, and *(b)* conducting several surveys in parallel.

A single query. One way to evaluate a single stratified sampling query is by applying reservoir sampling [16], which requires a single pass over the sampled dataset. It is well known how to conduct reservoir sampling on a single computer. However, such approach is unscalable and unsuitable for distributed datasets. Hence, our first goal is to provide an efficient and scalable distributed method for answering stratified sampling queries, without bias, using MapReduce.

Parallel surveys. Many companies, such as market-research companies, conduct multiple surveys in parallel. In parallel sampling, it is effective to share individuals among different surveys, to reduce costs. For instance, when user data should be anonymized before the delivery of the data to a costumer, verifying that there is no breach of privacy is expensive, thus, it is desired to share as many anonymized individuals as possible among the different surveys.

EXAMPLE 2. *A market research company needs to conduct 10 surveys for 10 different costumers, where each survey requires processing data of 1000 genuine*[1] *users of the social network. Without sharing, it is required to conduct 10,000 authenticity tests. Sharing may reduce the number of authenticity tests to 1000, and by that reduce costs.*

Yet, sometimes it is undesired to share individuals in different surveys. For example, when sharing the same individual in multiple samples, too much sensitive information about the individual may be disclosed, jeopardizing the anonymity of the individual or violating privacy restrictions. In online surveys, too much sharing may lead to survey fatigue. An optimal selection of individuals for surveys depends on the inter-dependencies between them. We refer to the problem of choosing the individuals for multiple parallel surveys under sharing constraints as *multi-survey stratified sampling*.

When processing a multi-survey stratified sampling query, there is a need to find a set of answers to a given set of stratified-sampling queries. The costs involved in conducting a survey may encourage sharing individuals in different surveys whereas the need to provide answers which are non-biased samples may restrict sharing. The following example illustrates such a scenario.

EXAMPLE 3. *Two surveys need to be conducted by sending questionnaires to members of an online social network (i.e., to individuals). The data of each individual must be properly anonymized by experts to avoid exposing sensitive personal information. One survey should be performed on a group of 50 men, and the other on a group of 100 singles. The cost of anonymizing a user is $1.*

By sharing as many individuals as possible among the surveys, it is possible to reduce anonymization costs. A simple way to maximize the sharing, is by choosing 50 single men to participate in both surveys and other 50 singles to participate in the second survey. The problem with this approach is that the 50 single men in the first answer do not provide a representative sample for the first survey. This is because in a random selection we do not expect all men to be singles.

[1]According to [2], 8.7% of Facebook users are fake.

In our approach we deal with the problem of Example 3 by calculating the number of single men that should be selected in a proper stratified sample for each part of the survey selection, i.e, for *(1)* merely the first survey, *(2)* merely the second survey, and *(3)* both surveys. Then, we randomly choose the individuals, accordingly.

Generalizing this approach to a full solution is intricate, because it requires examining which individuals should be shared, for every selection of strata from different surveys. Actually, this problem is NP-Hard. Hence, we do not expect to find a polynomial-time algorithm for the problem, and we settle for an efficient heuristic with a polynomial-time data complexity. Our approach is to formulate the problem of calculating the number of individuals for each combination of strata as a linear programming problem and apply random sampling based on these calculated values.

Contributions. Our main contributions are as follows.

- A framework for multi-survey stratified sampling over online social networks is provided. The framework allows formulating multiple stratified sampling queries and specifying constrains on the sharing of individuals in different surveys.
- We present an efficient and scalable distributed algorithm, for answering a single stratified-sampling query.
- We present an efficient distributed algorithm for answering multi-survey stratified-sampling queries, while minimizing the overall survey costs.

We present an experimental evaluation over a distributed system to illustrate the effectiveness of our algorithms.

2. RELATED WORK

Companies such as Gnip provide an API for acquiring data from online social networks such as Twitter, Facebook, and Foursquare. However, these APIs do not provide a means for acquiring stratified samples of the data. Several papers investigated sampling of online social networks [6, 10, 11], in a non-distributed environment.

The problem of optimally producing a simple random sample from k distributed streams of data has been studied in several papers (see [3, 15]). This is a generalization of the reservoir sampling approach to the case of handling multiple data streams from distributed sources. Their goal is to produce a simple sample in an unbiased manner, without the need to iterate over all the data. However, in stratified sampling the partition of the population into disjoint sets is only known when a sampling query is posed, and thus, typically it is different from the partition into clusters. Hence, it is impossible to simply apply a reservoir sampling algorithm on the different clusters to compute a stratified sample, so our algorithms cannot be built on top of their algorithms. Vojnovic et al. [17] study the problem of sampling with accuracy guarantees, but not for stratified sampling.

Frequently, a sample should be drawn from a subset of the population having a certain property, and a fixed-size random sample of the entire population will not suffice. Thus, Grover et al. [7] developed an extension of MapReduce for predicate-based sampling. However, this extension relies on an assumption that the data is stored in splits, where each split represents a random sample of the entire data. Otherwise, the resulting sample would be biased (see [12] for

further details). Specifically, this assumption does not hold in cases where data are not distributed randomly to the clusters, e.g., in the typical case where machines in a certain geographical region store data coming from this region.

In this paper, we introduce and study the novel problems of sharing individuals among surveys and processing stratified sampling queries in a distributed environment.

3. FRAMEWORK

In this section we define our framework, and we provide the syntax and semantics of stratified-sampling queries. To shorten the notation, we denote by $\overline{0,n}$ the set $\{0,1,\ldots,n\}$.

3.1 Dataset

Let $S = (P_1, P_2, \ldots, P_n)$ be a schema over the n properties (or attributes) $P_1, P_2, \ldots, P_n$. Let $D = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_n\}$ be the domains of the properties $P_1, P_2, \ldots, P_n$, respectively. A dataset is a set of *individuals*, denoted R, where each individual is represented by a tuple $(p_1, p_2, \ldots, p_n) \in \mathbf{D}_1 \times \mathbf{D}_2 \times \cdots \times \mathbf{D}_n$. In correspondence with the terminology of statisticians, we sometimes refer to a set of individuals as a *population*. Our model can be easily modified to deal with multisets rather than sets, however, we avoid this to simplify the notations. Note that properties may relate to edges of the network, such as the existence of a specific edge or the number of neighbors of an individual.

3.2 Sampling Queries

Sampling queries define the selection of individuals from a population in a *survey design*. We refer to surveys, in this context, as statistical surveys which are used to collect information from a sample of individuals in a systematic manner. A survey design is composed of a population, a *sample design* and a method for extracting information from an individual (e.g. an interview) [8]. The sample design is concerned with how to select, based on the properties of individuals, the part of the population to be included in the survey. We focus on a *stratified sample design* [1, 5], where the population is classified into subpopulations, also called strata, according to the properties of the individuals, and simple random samples (without replacement) are then drawn from each strata.

3.2.1 A Single Survey

A stratified sample design comprises strata, specified by a *stratified sampling design query*—SSD query, for short. An SSD query is essentially a set of *stratum constraints* where each constraint defines a stratum. Formally, a stratum constraint has the form $s_k = (\varphi_k, f_k)$, where φ_k is a propositional formula and $f_k \in \mathbb{N}$ is the required *sample frequency*, i.e., the number of individuals to select from the stratum. The condition φ_k is a propositional formula in the style of the formulas of domain relational calculus (DRC), using the logical operators $\wedge$ (conjunction), $\vee$ (disjunction) and $\neg$ (negation). The expression $\sigma_\varphi(R)$ is produced by applying the selection operator on the set of individuals R, with φ as the selection condition. For instance, a stratum constraint to define a sample of 50 individuals that are either men with income lower than 50000 or women with income greater than 100000 is formulated as follows:

$$s_k = \big((\text{gender} = \text{male} \wedge \text{yearly\_income} < 50000) \vee (\text{gender} = \text{female} \wedge \text{yearly\_income} > 100000), 50 \big)$$

A stratum constraint specifies a single stratum. An SSD query is a set of stratum constraints and it is denoted $Q = \{s_1, s_2, \ldots, s_m\}$. For an SSD query Q to be *valid*, the strata defined by every pair of stratum constraints φ_{k_1} and φ_{k_2} must be disjoint, i.e. $\sigma_{\varphi_{k_1}}(R) \cap \sigma_{\varphi_{k_2}}(R) = \emptyset$.[2]

We now define the semantics of SSD queries. Let $Q = \{s_1, s_2, \ldots, s_m\}$ be an SSD query. We say that a tuple $t \in R$ *satisfies* the stratum constraint $s_k = (\varphi_k, f_k)$ if t is in the selection of φ_k over the dataset of individuals R, i.e., if $t \in \sigma_{\varphi_k}(R)$. A subset $A_k \subseteq R$ *satisfies* the stratum constraint s_k if (1) there are exactly f_k tuples in A_k, and (2) all the tuples of A_k satisfy φ_k. A subset $A \subseteq R$ satisfies the SSD query Q if $A = \cup_{k=1}^{m} A_k$ such that for each $1 \leq k \leq m$, A_k satisfies s_k. Note the requirement of avoiding surplus tuples in A, i.e., $|A| = \sum_{k=1}^{m} f_k$.

A stratum constraint s_k is *satisfiable* over R if there exists a subset $A_k \subseteq R$ that satisfies s_k. Similarly, an SSD query Q is *satisfiable* over R if all its stratum constraints are satisfiable over R.

We want the selection of tuples to be a statistically valid sample. Accordingly, we say that a subset $A_k \subseteq R$ is a *representative sample* with respect to the stratum constraint s_k, if $\sigma_{\varphi_k}(A_k)$ represents a simple random sample of $\sigma_{\varphi_k}(R)$. We say that $A \subseteq R$ is an *answer* to Q, if A is a union of m sets $A = \cup_{k=1}^{m} A_k$, such that each set A_k (1) satisfies s_k, and (2) is a representative sample with respect to s_k.

3.2.2 Multiple Surveys

We now formalize the problem of conducting multiple surveys in parallel, while minimizing the overall expenses. We express this problem using a *multi stratified-sample design query* (MSSD query, for short). The input consists of (1) a set of sample designs, (2) the cost of collecting information from an individual (e.g., interview costs, anonymization costs, etc.), and (3) the cost of sharing an individual in multiple surveys. In this setting, each survey (sample design) is formulated as an SSD query. We denote the set of queries by $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_n\}$.

The *interview cost* c_i of an SSD query Q_i is the cost[3] of collecting information from a single individual, merely for Q_i. When sharing individuals in surveys, the cost is defined with respect to a set of surveys. A set $\tau \subseteq \overline{0,n}$ defines a subset of $\mathcal{Q}$, e.g., $\tau = \{1,3\}$ defines the subset $\{Q_1, Q_3\}$. The *shared survey cost* c_τ is the cost of sharing an individual by exactly those surveys whose index is in τ.

When the shared survey cost c_τ is the sum of the interview costs of the queries defined by τ, this reflects indifference to sharing. To simplify the formulation of MSSDs, we define indifference to sharing as the default sharing cost dc_τ, for every subset τ, i.e., $dc_\tau = \sum_{i \in \tau} c_i$. The following example illustrates a simple MSSD query in which the shared survey cost is different from the default value.

EXAMPLE 4. *Consider a query set $\mathcal{Q} = \{Q_1, Q_2\}$. Suppose that the first survey is a face-to-face survey with an interview cost $c_1 = \$20$, and the second survey is a telephone survey, with an interview cost $c_2 = \$4$. The cost of surveying an individual that is assigned to both surveys, $c_{\{1,2\}}$, is*

[2] Frequently, definitions of stratified sampling require the union of the strata to contain the entire population. We do not require this, as a generalization, however, adding this requirement does not change any of our results.

[3] *Cost* refers to expenses and not to computational costs.

$max(c_1, c_2) = \$20$. *This can be done only if conducting the interview of the second survey during the face-to-face meeting does not affect the statistical validity of the results and does not add expenses.*

To define the semantics of an MSSD query, let $(\mathcal{Q}, C)$ be an MSSD query where $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_n\}$ is a set of SSD queries and $C = \{c_\tau \mid \tau \subseteq \overline{1,n}\}$ is a set of shared survey costs. The answer set $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ is an answer to $(\mathcal{Q}, C)$ if for every $i \in \overline{1,n}$, A_i is an answer to Q_i. We denote by $union(\mathcal{A})$ the union of $A_1, \ldots, A_n$.

Given an individual t in $union(\mathcal{A})$, the SSDs of t are the answer sets that contain t, and they are represented by a subset $\tau(t) \subseteq \overline{1,n}$. For example, if t appears only in A_2 and in A_5, then $\tau(t) = \{2, 5\}$. The shared cost of t is defined according to $\tau(t)$. It is c_τ if $c_\tau \in C$, i.e., if this shared cost is defined by the given MSSD query, and it is the default cost dc_τ, otherwise. The cost of the entire answer set $\mathcal{A}$ is the sum of the costs of all the tuples in the sets of $\mathcal{A}$, i.e., $c_\tau(\mathcal{A}) = \sum_{t \in union(\mathcal{A})} c_{\tau(t)}$.

Choosing individuals when answering a given MSSD query is conducted in a probabilistic manner. Thus, an algorithm to answer $(\mathcal{Q}, C)$ is expected to produce different answers in different runs—answers that may have different costs. Thus, we cannot compare two algorithms based on a single run.

Suppose ALG_1 and ALG_2 are evaluation algorithms for MSSD queries. We denote by $\mathcal{A}_i^1$ the answer to $(\mathcal{Q}, C)$ in the i-th run of ALG_1, and by $\mathcal{A}_i^2$ the answer in the i-th run of ALG_2. We say that ALG_1 is *at least as effective as* ALG_2, on $(\mathcal{Q}, C)$, on the average, if the costs of the answers of ALG_1 are not greater than the costs of the answers of ALG_2. That is, the difference between the costs of the answers of ALG_2 to the costs of the answers of ALG_1 is non-negative when the number of runs K approaches infinity:

$$\lim_{K \to \infty} \frac{1}{K} \left(\sum_{i=1}^{K} \left(c(\mathcal{A}_i^2) - c(\mathcal{A}_i^1) \right) \right) \geq 0$$

An algorithm to answer MSSD queries is *optimal* if it is at least as effective, on every MSSD $(\mathcal{Q}, C)$, as any other algorithm that answers MSSD queries.

3.2.3 Problem Definition

In a distributed environment, the dataset R is stored on several machine such that each machine can execute queries over the tuples it stores or send tuples to other machines. Our goals are: *(1)* to answer SSD queries in a distributed environment, *(2)* to provide an efficient optimal algorithm to answer MSSD queries, and *(3)* to answer MSSD queries in a distributed environment.

4. ANSWERING AN SSD QUERY

In this section we present our algorithm for answering a single SSD query $Q = \{s_1, \ldots, s_m\}$ over a set R of individuals. Throughout this section, we use the notations of Section 3. First, we discuss sequential methods for evaluating a single SSD query, and then we present an algorithm that is built for the MapReduce framework, to provide a scalable distributed method for processing SSD queries.

4.1 Sequential Sampling

One way to answer the query Q is by choosing a simple random sample of f_k individuals from stratum $\sigma_{\varphi_k}(R)$, for each $1 \leq k \leq m$, and then taking the union of the m samples as the answer to Q. In the worst case, the number of stratum constraints can be exponential in the number of attributes of R, and hence, this approach may be impractical.

A *reservoir algorithm* is an algorithm that in a single sequential pass over R chooses the tuples of the sample [16]. A particular reservoir algorithm is *Algorithm R*, attributed to Alan Waterman. Algorithm R scans the given relation sequentially and constructs a *reservoir* that at the end of the scan contains the sample. Let t_{i+1} be the $(i+1)$-st tuple processed by the algorithm. If $i < f_k$, then t_{i+1} is added to the reservoir. Otherwise, with probability of $f_k/(i+1)$ the tuple t_{i+1} replaces a tuple that is in the reservoir. The replaced tuple is chosen uniformly among the f_k tuples already in the reservoir. Consequently, the reservoir holds a simple random sample of the processed tuples, at any step of the scan during the run.

Using a reservoir algorithm allows selecting the sample while minimizing the number of I/O requests. However, to make the sampling scalable and to support sampling of data that are stored on many machines, the sampling should be conducted as a distributed process.

4.2 Distributed Sampling

When the data is distributed among multiple machines a stratified sample can be produced using a distributed algorithm. As an example, consider an SSD query that specifies a requirement for two men and three women. At first glance producing the sample may seem an easy task. Each machine can produce an intermediate sample of males and females and, at the end, unify these samples. This, however, will not guarantee that the final sample will include exactly two males and three females. To overcome this, each machine can produce an intermediate sample of two males and three females and have the unification produce a final uniform sample of two males and three females from all the intermediate samples. This approach is simple and efficient, yet it may produce a biased sample which would be statistically invalid. To illustrate this point, consider a scenario in which the unification should select two males from two intermediate samples—one sample S_1 comprises two males that were selected out of four males, and another, S_2, comprises two males that were selected from a set of eight males. In S_1, the probability of a male to be selected for the intermediate sample is $1/2$, and in S_2 it is $1/4$. Since the overall number of males from which these two intermediate samples were drawn is 12, each male should have a probability of $1/6$ to be selected for the final sample. However, when selecting uniformly a random sample of two males from S_1 and S_2, for a man in S_1 there is a probability of $1/4$ to be in the final sample, whereas for a man in S_2 the probability to be in the final sample is $1/8$. So, to achieve a final sample which is truly unbiased and uniform, the selection from each intermediate sample must be adjusted to account for the probability of each male to appear in the intermediate sample. In this case, males from S_1 should be selected with probability $1/3$ and those from S_2, should be selected with probability $2/3$.

We now discuss our distributed algorithm for answering SSD queries. This algorithm is designed as a MapReduce program. MapReduce is a programming framework that allows processing and generating large data sets [4]. A MapReduce program specifies *(1)* a map function that processes key-value pairs to generate a set of intermediate key-value

$$\text{map}(null, t) \rightarrow [(s_k, t)] \qquad (\textit{if } t \textit{ satisfies } s_k)$$
$$\text{reduce}(s_k, [t_1, \ldots, t_N]) \rightarrow (\text{SRS}([t_1, \ldots, t_N], f_k))$$

Figure 1: Naive processing of a single SSD query.

pairs, and *(2)* a reduce function that merges all intermediate values associated with the same intermediate key.

4.2.1 Naive Map-Reduce Sampling

First, we present a naive algorithm, and then we will show how to improve it. In this algorithm, we simultaneously draw a simple random sample from each stratum. The mapping phase partitions the tuples according to their matching stratum constraints, and the reduce phase generates the samples from the partitions.

The MapReduce program is as follows. Given a tuple t, if t satisfies stratum constraint s_k, then t is mapped to the pair (s_k, t). In this pair, s_k is the key and t is the value. A tuple that does not satisfy any stratum constraint is ignored. Since stratum constraints are disjoint, any tuple can satisfy at most a single stratum constraint.

Each reduce function receives a stratum constraint, of the form $s_k = (\varphi_k, f_k)$, and the list of tuples $[t_1, \ldots, t_K]$ that were mapped to s_k. The reduce function produces a list containing a sample of f_k tuples, uniformly selected from the list $[t_1, \ldots, t_K]$. Note that if $f_k > \sum_{i=1}^{K} |\bar{S}_i|$ (i.e., there are not enough tuples in the input to draw the sample), then all the tuples in the list are selected. Figure 1 depicts this program, where *SRS* refers to the selection of a simple random sample, e.g., using Algorithm R.

4.2.2 Improved Map-Reduce Sampling (MR-SQE)

In the naive approach, any tuple that satisfies a stratum constraint is sent over the network to the appropriate reduce function, however, only a sample of these tuples is required. For each stratum, the sample selection is done synchronously since a single reduce function is applied per stratum. We can increase concurrency and reduce the amount of data transmitted over the network by using a combiner operation [4]. The combiner performs a partial selection of the tuples produced in the map phase. It is conducted before tuples are sent over the network. Basically, the combiner locally selects a sample of the tuples generated during the map phase, on each machine that performs a map task, using a reservoir algorithm (Algorithm R). Note that the combiner provides an intermediate sample, not a final one.

When using a combiner to select local samples, the reduce operation receives intermediate samples and produces the final sample. For the chosen sample to be valid, it must constitute a simple random sample, in the sense that every subset of tuples of equal size should be given an equal probability of being chosen.

The reduce function receives a list of samples and for each sample, needs to be aware of the overall number of tuples from which the sample was drawn. Hence, each intermediate sample produced by the combiner has the form $S = (\bar{S}, \bar{N})$, where $\bar{S}$ is the intermediate sample and $\bar{N}$ is the size of the set from which $\bar{S}$ was drawn. This approach, referred to as *Map Reduce Single Query Evaluator* (MR-SQE), is depicted in Figure 2. MR-SQE consists of *(1)* a map function similar to that of the naive algorithm, *(2)* a combiner function

$$\text{map}(null, t) \rightarrow [(s_k, t)] \qquad (\textit{if } t \textit{ satisfies } s_k)$$
$$\text{combine}(s_k, [t_1, \ldots, t_N]) \rightarrow (\text{SRS}([t_1, \ldots, t_N], f_k), N)$$
$$\text{reduce}(s_k, [(\bar{S}_1, \bar{N}_1), \ldots, (\bar{S}_K, \bar{N}_K)]) \rightarrow$$
$$[\textit{unified-sampler}(\{(\bar{S}_1, \bar{N}_1), \ldots, (\bar{S}_K, \bar{N}_K)\}, f_k)]$$

Figure 2: MR-SQE: processing a single SSD query.

Algorithm 1 unified-sampler (implements an S-ALG)

unified-sampler$(\{(\bar{S}_1, N_1), \ldots, (\bar{S}_K, N_K)\}, n)$

Input 1: $\{(\bar{S}_1, N_1), \ldots, (\bar{S}_K, N_K)\}$ – *(intermediate)*
Input 2: n – *(the required sample size)*
Output: The result sample $\bar{S}$

1: **if** $\sum_{i=1}^{K} |\bar{S}_i| < n$ **then**
2: **return** $\bigcup_{i=1}^{K} \bar{S}_i$
3: $N \leftarrow \sum_{i=1}^{K} N_i$
4: $I \leftarrow$ *uniformly selected* n *indexes from* $\overline{1, N}$
5: $\bar{S} \leftarrow \emptyset$
6: $L \leftarrow 1$
7: $U \leftarrow N_1$
8: **for** $i = 1$ to K **do**
9: $c \leftarrow |I \cap \overline{L, U}|$
10: $\bar{S}' \leftarrow$ *uniformly selected* c *tuples from* $\bar{S}_i$
11: *(* Comment: in Line 10,* $c \leq |\bar{S}_i|$ **)*
12: $\bar{S} \leftarrow \bar{S} \cup \bar{S}'$
13: $L \leftarrow L + N_i$
14: $U \leftarrow U + N_{i+1}$, if $i < K$
15: **return** $\bar{S}$

that draws samples using Algorithm R and outputs each intermediate sample with the size of the set from which it was drawn, and *(3)* a reduce function which selects the final sample from the intermediate samples of each stratum.

The reduce function applies the *unified sampler* algorithm, depicted as Algorithm 1. This algorithm produces the final sample from a set of intermediate samples generated by the combiner functions. Note that selection by the reduce function is over intermediate samples that have all been drawn from tuples that match a single stratum constraint.

Algorithm 1 receives K intermediate samples $\bar{S}_1, \ldots, \bar{S}_K$. Let $R_1, \ldots, R_K$ denote the sets from which $\bar{S}_1, \ldots, \bar{S}_K$ were drawn, respectively. Recall that there are exactly N_i tuples in R_i, for each $i \in \overline{1, K}$ and that $R = \cup_{i=1}^{K} R_i$, where R is the entire set of individuals. Algorithm 1 is invoked by the reducer on samples consisting of tuples that match some stratum constraint s_k. Hence, the task of Algorithm 1 is selecting n tuples in an unbiased manner—it is not required to filter out tuples that do not match the stratum constraint.

The algorithm begins by checking if there are enough tuples to draw a sample of size n. If not, it merely unifies the intermediate samples. If there are more than n tuples, then I in Line 4 represents a virtual selection from the entire set R and by that helps determining how many tuples will be selected from each $\bar{S}_i$, in a way that takes into account the sizes $N_1, \ldots, N_K$. This generates for each set $\bar{S}_i$ a value c that is probabilistically proportional to N_i. Thus, the algorithm iterates over the intermediate samples $\bar{S}_1, \ldots, \bar{S}_K$, and uniformly draws a sample of c tuples from each $\bar{S}_i$. Then, the samples are unified to form the result sample.

The selection of c tuples from $\bar{S}_i$ in Line 10 is conducted as a selection without replacement. In each draw, a random tuple of $\bar{S}_i$ is selected and removed from the set. Suppose there are m tuples in $\bar{S}_i$, then in the i-th draw ($1 \leq i \leq c$), each tuple has a probability of $\frac{1}{m-i+1}$ to be selected. Notice that for each tuple of $\bar{S}_i$, its probability to be selected in c draws is $1-(1-\frac{1}{m})(1-\frac{1}{m-1})\cdots(1-\frac{1}{m-c+1})$, that is, $1-(\frac{m-1}{m})(\frac{m-2}{m-1})\cdots(\frac{m-c}{m-c+1}) = 1-\frac{m-c}{m} = \frac{c}{m}$.

EXAMPLE 5. *Consider a dataset R with 64 individuals—30 men and 34 women—distributed on two machines, such that R_1 comprises 20 men and 16 women; and R_2 consists of 10 men and 18 women. Also, consider two stratum constraints, s_1 and s_2, which require selecting 5 men and 6 women, respectively. Initially, a mapper process partitions R_1 into 20 men and 16 women. Then, a combiner process selects 5 random men from the 20 men. Another combiner process selects 6 random women from the 16 women. In parallel, another mapper process partitions R_2 into 10 men and 18 women, and combiner processes randomly select 5 men and 6 women from these sets. There are two reduce processes—one for s_1 and one for s_2. The reducer for s_1 receives 5 tuples (i.e. 5 men) from each combiner and selects from them the 5 men of the result. It does so by applying Algorithm 1 on $(\bar{S}_1, N_1), (\bar{S}_2, N_2)$, where $\bar{S}_1$ and $\bar{S}_2$ are the sets of 5 men, $N_1 = 20$ and $N_2 = 10$.*

Algorithm 1 proceeds as follows. In Line 3, it randomly selects 5 index numbers out of the range $\overline{1,30}$, say $I = \{1,3,10,22,28\}$. In the first iteration of the loop (Line 8), $I \cap \overline{L,U} = \{1,3,10,22,28\} \cap \overline{1,20} = \{1,3,10\}$. There are three numbers in $\{1,3,10\}$, therefore the algorithm selects three random men from S_1. In the next iteration, $I \cap \overline{L,U} = \{1,3,10,22,28\} \cap \overline{20,30} = \{22,28\}$. There are two numbers in $\{22,28\}$, so two men are randomly selected from S_2. The five randomly selected men are returned in Line 15.

4.2.3 Correctness

To show that Algorithm 1 is correct, we need to show that it is equivalent to a non-distributed sampling algorithm. We begin by showing that the algorithm draws the required number of tuples, i.e., $\min(\sum_{i=1}^{K}|\bar{S}_i|, n)$ tuples. First, we show that the comment in Line 11 holds, so that indeed there are enough tuples in each $\bar{S}_i$ to select c tuples in iteration i. There are two cases to consider. Case 1, $\bar{S}_i$ contains n tuples. Then, $|\bar{S}_i| = |I| \geq |I \cap \overline{L,U}| = c$. Case 2, $\bar{S}_i$ contains less than n tuples. Then, all the tuples of the initial set were selected by the combiner, so $|\bar{S}_i| = N_i$. Now, $c = |I \cap \overline{L,U}| \leq |\overline{L,U}| = N_i = |\bar{S}_i|$. Thus, in both cases, $|\bar{S}_i| \geq c$. Secondly, the sum of the values of c over K iterations, is $|I|$, which is equal to n, so, the claim holds.

We now show, inductively, that the algorithm produces a simple random sample. The induction is over the size n of the final sample. We need to show that for every n, every subset of R of size n has the same probability to be selected.

Suppose $n = 1$. There are two cases. In Case 1, we compare two samples of a single tuple where in both samples the tuple is initially selected from R_i, for some $1 \leq i \leq K$. Since tuples of R_i are selected uniformly by the combine function, these two samples have the same probability to be selected. In Case 2, one sample comprises a single tuple of R_i and the other comprises a single tuple of R_j, $1 \leq i < j \leq K$. The probability of the first sample is $\frac{|R_i|}{|R|} \cdot \frac{1}{|R_i|} = \frac{1}{|R|}$ because there is a probability of $\frac{|R_i|}{|R|}$ to choose a tuple from R_i when drawing the indexes of I, and a probability of $\frac{1}{|R_i|}$ to select a specific tuple from R_i. Similarly, when replacing R_i by R_j, the probability of the second sample is also $\frac{1}{|R|}$. Hence, the two samples have the same probability.

We assume the following hypothesis: every sample of size n has the same probability to be selected. Now, given a sample σ of size $n+1$, let t be the last tuple that is added to the result sample. The probability to select σ is the probability to select $\sigma - \{t\}$ (the n tuples selected before t) multiplied by the probability to select t. Suppose t is selected from R'_i, where R'_i is the result of removing from R_i the tuples that were already selected for σ. Then, the probability to select t is $\frac{|R'_i|}{|R|-n} \cdot \frac{1}{|R'_i|}$. Hence, for every sample of size $n+1$ there is the same probability to select it: the probability to select a sample of size n (equal for any sample of size n, according to the hypothesis) multiplied by $\frac{1}{|R|-n}$, which is independent of t. Thus, the hypothesis holds for a sample of size $n+1$.

REMARK .1. *Consider some sub-relation R_j. Let r be the number of tuples in R_j, prior to the mapping phase, s_k be some stratum constraint and $\bar{S}'$ be the set of those c tuples selected from R_j by the algorithm. Then, the probability of finding y tuples of $\bar{S}'$ among the first x tuples of R_j is $\binom{c}{y}\binom{r-c}{x-y}/\binom{r}{x}$, because there are $\binom{c}{y}$ options to choose y tuples from $\bar{S}'$; there are $\binom{r-c}{x-y}$ options to choose completing $x-y$ tuples from the $r-c$ tuples of $R_j \setminus \bar{S}'$, i.e., from the tuples that are not selected in the sample; and there are $\binom{r}{x}$ options to choose x tuples from the r tuples of R_j, altogether. This probability mass function yields a hypergeometric distribution for the values of y [5].*

5. ANSWERING AN MSSD QUERY

In this section we present algorithms for answering MSSD queries, while aiming to minimize survey costs. First, we present Algorithm MR-MQE which is an extension of MR-SQE. Algorithm MR-MQE can evaluate MSSD queries efficiently, but does so while ignoring survey costs. We present an MSSD evaluation algorithm called CPS, which is based on integer programming, and we show that CPS is optimal. Then, we present an algorithm called MR-CPS which is a relaxation of CPS and has linear-time data complexity. Finally, we show how MR-CPS can be implemented as a series of MapReduce programs.

Throughout this section we use the terminology and notations of Section 3. We represent a given MSSD query as a pair $(\mathcal{Q}, C)$, where $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_n\}$ is a set of n SSDs and C is a set of shared survey costs.

5.1 Algorithm MR-MQE

Before addressing the issue of finding an optimal answer to an MSSD query, we consider the problem of answering multiple SSD queries (or answering an MSSD query without taking into account the survey costs). This would serve as a benchmark for comparing to it the optimal algorithm.

One way to compute an answer to $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_n\}$, is to independently run MR-SQE for each $Q_i \in \mathcal{Q}$. This approach, however, requires n passes over R, and each pass requires many I/O operations. A more efficient approach is to slightly alter Algorithm MR-SQE. Since the same stratum constraint can appear in several SSDs, we use, as mapping keys, pairs (Q_i, s_k), where $s_k \in Q_i$, instead of using merely s_k. Accordingly, in the map phase, instead of creating a list

with a single pair (s_k, S), we create a list containing all the pairs $((Q_i, s_k), (\{t\}, 1))$ such that $s_k \in Q_i$ and the tuple t satisfies s_k. The reduce remains the same. Since running this algorithm is semantically equivalent to running MR-SQE for each SSD, it follows that this algorithm produces an answer to $\mathcal{Q}$. We refer to this modified version of MR-SQE as Algorithm *MR-MQE*.

5.2 Optimally Answering MSSD Queries

While finding some answer to an MSSD query can be done efficiently, the problem of finding an optimal answer is NP-Hard. We show this by reducing to it the minimum vertex cover problem, which is known to be NP-Complete. Given a graph $G = (V, E)$, we create a population in which the vertices are the individuals (i.e., the tuples), and there is an SSD with a single stratum constraint for each edge. For every edge $e = \{v, u\}$, the stratum constraint contains a propositional formula that requires the individual to either be v or u and the sample frequency is exactly 1. (Note that the constraints may degenerate the sampling causing the result to be a strict selection.) The interview cost of any SSD is 1 and sharing has no cost. The optimal answer to this MSSD is a selection of individuals that is equivalent to a minimum vertex cover. Hence, the problem is NP-Hard.

There is another apparent difficulty. An optimal answer must be a representative sample of each stratum constraint, while the selection should minimize the survey cost. As discussed in Section 1, this may seem contradictory, because controlling the selection could lead to a biased sample. Next, we show how to deal with that.

5.2.1 An Outline of the Approach

We begin with an example that illustrates the conflict in the selection of the sample and the general approach.

EXAMPLE 6. *Consider an MSSD query $\mathcal{Q} = \{Q_1, Q_2\}$, where $Q_1 = (s_{1,1}, s_{1,2})$, $Q_2 = (s_{2,1}, s_{2,2})$ and let the stratum constraints be as specified in the following table.*

$s_{1,1} = (gender{=}male, 10)$	$s_{1,2} = (gender{=}female, 15)$
$s_{2,1} = (income{<}50000, 12)$	$s_{2,2} = (income{>}200000, 12)$

Suppose the interview cost for both SSDs is one dollar and the shared cost is also a dollar, i.e., $C = \{c_{\{1\}}, c_{\{2\}}, c_{\{1,2\}}\}$, where $c_{\{1\}} = c_{\{2\}} = c_{\{1,2\}} = 1$. How should one construct an optimal answer? Obviously, sharing in this case reduces costs. A woman with income above \$200000 satisfies both $s_{1,2}$ and $s_{2,2}$, so such a woman can participate in two surveys. Thus, we could try choosing 12 women with income greater than \$200000, 3 women with income below \$50000, and 10 men with income below \$50000. It is easy to see that such selection minimizes costs, but it is unlikely that such selection will provide a representative sample.

In Example 6, the number of women with income greater than \$200000 was manipulated to increase sharing. Such bias damages the representativeness of the sample. Yet, how can we know how many individuals to select for each combination of stratum constraints? The solution is to apply a two-step process. In the first step, some (non-optimal) answer is computed (e.g., using MR-MQE). We count the number of women with income greater than \$200000 in the answers produced for Q_1 and Q_2. Note that now this number is based on a representative sample. Then, in the second step, we uniformly select the required number of women with income greater than \$200000 and share as many of them as possible. We do the same for other combinations of constraints, e.g., for men with income below \$50000, etc.

5.2.2 Preliminary Definitions and Notations

Before presenting the algorithm, we provide necessary notations and definitions. Given a set of SSD queries $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$, a *stratum selection* is a set of stratum constraints with at most one stratum constraint for each SSD of $\mathcal{Q}$. In Example 6, $\{s_{1,2}, s_{2,2}\}$ and $\{s_{2,2}\}$ are stratum selections, however, $\{s_{1,1}, s_{1,2}\}$ is not a stratum selection because $s_{1,1}$ and $s_{1,2}$ both belong to the same query Q_1. We denote by $[[\mathcal{Q}]]$ the set of all possible stratum selections over $\mathcal{Q}$.

The *propositional projection* of a stratum selection σ on query Q_i is denoted by $\pi_i(\sigma)$. When Q_i has a stratum constraint in σ, it is defined as the propositional formula of this stratum constraint. Otherwise, it is the negation of the disjunction of the stratum constraints of Q_i.

$$\pi_i(\sigma) = \left\{ \begin{array}{cc} \varphi_{i,j} & \text{if } \exists s_{i,j} \in \sigma \\ \neg(\varphi_{i,1} \vee \varphi_{i,2} \vee \ldots \vee \varphi_{i,n}) & \text{otherwise} \end{array} \right\}$$

That is, if there is a stratum constraint $s_{i,j}$ of query Q_i in σ, then the projection is the condition that defines $s_{i,j}$. So, for Example 6, the projection $\pi_2(\{s_{1,2}, s_{2,2}\})$ is the condition of $s_{2,2}$, which is $income > 200000$. If there is no stratum constraint of Q_i in σ then the projection is a condition that is satisfied only by individuals that do not satisfy any condition of Q_i. In Example 6, for instance, the projection $\pi_2(\{s_{1,1}\})$ is $\neg(income < 50000 \vee income > 200000)$.

A tuple of R *satisfies* a stratum selection σ if it satisfies $\pi_i(\sigma)$ for every $1 \leq i \leq n$. That is, the tuple satisfies σ only if it satisfies all the propositional formulas of the stratum constraints in σ without satisfying any other propositional formula. For instance, in Example 6, for $\sigma = \{s_{1,1}\}$, the condition $\pi_1(\sigma) \wedge \pi_2(\sigma)$ represents a stratum of men with an income between \$50000 and \$200000.

The *SSD indexes* of a stratum selection σ, denoted by $I(\sigma)$, is the set of all the indexes of the SSDs that have a stratum constraint in σ. For example, $I(\{s_{1,1}\}) = \{1\}$ and $I(\{s_{1,1}, s_{2,1}\}) = \{1, 2\}$.

Let $\{A_1, A_2, \ldots, A_n\}$ be an answer to an MSSD $\mathcal{Q}$. Given a stratum selection σ and an index $i \in I(\sigma)$, the *stratum-selection frequency* over answer A_i, denoted by $F(A_i, \sigma)$, is the number of tuples in A_i that satisfy the stratum selection σ. For instance, in Example 6, $F(A_1, \{s_{1,1}\}) = 5$ indicates that there are exactly 5 men with an income between \$50000 and \$200000 in A_1. Similarly, a *stratum selection limit* over the dataset R, denoted by $L(\sigma)$, is the number of tuples of R that satisfy the stratum selection σ, i.e., $L(\sigma) = F(R, \sigma)$.

5.2.3 Algorithm CPS

We now present the *Constraint Program Selector* algorithm, CPS for short. The algorithm receives an MSSD query $(\mathcal{Q}, C)$. First, an initial representative non-optimal answer $\mathcal{A} = \{A_1, \ldots, A_n\}$ is computed for $\mathcal{Q}$. This initial answer provides the stratum selection frequencies $F(A_i, \sigma)$, for every $\sigma \in [[\mathcal{Q}]]$ and $i \in \overline{1, n}$. These frequencies are used to set the stratum selection frequencies of the answer, to be as in the representative answer. Then, an integer program is constructed and solved to find the optimal selection.

To define the integer program, we create an integer decision variable $X_\tau(\sigma)$ for every $\sigma \in [[\mathcal{Q}]]$ and every $\tau \subseteq I(\sigma)$. Ultimately, a value is assigned to each variable. The value

Decisions	$\{X_\tau(\sigma) \mid \sigma \in [[\mathcal{Q}]], \tau \subseteq I(\sigma)\}$
Domains	$\forall \sigma \in [[\mathcal{Q}]], \forall \tau \subseteq I(\sigma) : X_\tau(\sigma) \in \overline{0,\infty}$
Equivalence constraints	$\forall i \in \overline{1,n}, \forall \sigma \in [[\mathcal{Q}]] :$ $\sum_{\tau \subseteq I(\sigma) \text{ where } i \in \tau} X_\tau(\sigma) = F(A_i, \sigma)$
Upper bound constraints	$\forall \sigma \in [[\mathcal{Q}]] :$ $\sum_{\tau \subseteq I(\sigma)} X_\tau(\sigma) \leq L(\sigma)$
Objective	minimize $\left(\sum_{\sigma \in [[\mathcal{Q}]]} \sum_{\tau \subseteq I(\sigma)} c_\tau \cdot X_\tau(\sigma)\right)$

Figure 3: The Integer Program (IP)

assigned to the variable $X_\tau(\sigma)$ specifies the number of tuples that should *(1)* be drawn from those tuples of R that satisfy σ; and *(2)* be included in the answers of $\mathcal{A}$ whose indexes are in τ. For instance, in Example 6, the expression $X_{\{1\}}(\{s_{1,1}\}) = 5$ specifies that 5 men with an income between \$50000 and \$200000 should be selected to be in A_1.

The question of finding an optimal answer to $\mathcal{Q}$ can now be formulated as an integer programing problem whose goal is to find the optimal assignment to each variable. For every $\sigma \in [[\mathcal{Q}]]$ and every $i \in \overline{1,n}$, we define a constraint that assures the sum of all relevant decision variables is equal to $F(A_i, \sigma)$. For every $\sigma \in [[\mathcal{Q}]]$, there is a constraint that limits the sum of all the relevant decision variables not to exceed $L(\sigma)$. The integer program is depicted in Figure 3.

After solving the integer program problem and determining the value of each decision variable, we create a single SSD query that contains a stratum constraint for each stratum selection $\sigma \in [[\mathcal{Q}]]$. The propositional formula of the constraint is $\varphi(\sigma) = \pi_1(\sigma) \wedge \pi_2(\sigma) \cdots \wedge \pi_n(\sigma)$. The required sample frequency is $f(\sigma) = \sum_{\tau \subseteq I(\sigma)} X_\tau(\sigma)$. Thus, the stratum constraint that corresponds to σ is $s(\sigma) = (\varphi(\sigma), f(\sigma))$. In Example 6, for $\sigma = \{s_{1,1}\}$, the value of $f(\sigma)$ is equal to the number of men with an income between \$50000 and \$200000 to be selected for A_1.

The stratum constraints created for the stratum selections of $[[\mathcal{Q}]]$ yield an SSD query Q'. We use some sampling algorithm to answer Q'. This provides a *combined answer*, denoted by A', comprising $\sum_{\sigma \in [[\mathcal{Q}]]} \sum_{\tau \subseteq I(\sigma)} X_\tau(\sigma)$ tuples.

Next, we create a set $\mathcal{A}^* = \{A_1^*, A_2^*, \ldots, A_n^*\}$ of empty sets. For each $\sigma \in [[\mathcal{Q}]]$ along with every $\tau \subseteq I(\sigma)$, we arbitrarily extract $X_\tau(\sigma)$ tuples that satisfy σ from A'. We add each of these tuples to the sets of $\mathcal{A}^*$ whose indexes are contained in τ. When this process completes, A' is empty and $\mathcal{A}^*$ represents the answer of CPS to $\mathcal{Q}$. The pseudocode of CPS is presented as Algorithm 2.

5.2.4 Correctness of CPS

We now show the correctness of Algorithm 2. We begin by showing that $\mathcal{A}^*$ satisfies $\mathcal{Q}$. Let $s_{i,k} = (\varphi_{i,k}, f_{i,k})$ be an arbitrary stratum constraint of Q_i. When CPS terminates, in A_i^* there are $\sum_{\sigma \in [[\mathcal{Q}]] \wedge s_{i,k} \in \sigma} \sum_{\tau \subseteq I(\sigma) \wedge i \in \tau} X_\tau(\sigma)$ tuples that satisfy $s_{i,k}$. According to the equivalence constraints of the IP, this is equal to $\sum_{\sigma \in [[\mathcal{Q}]] \wedge s_{i,k} \in \sigma} F(A_i, \sigma)$, which is exactly the number of tuples that satisfy $s_{i,k}$ in the initial answer A_i obtained by the non-optimal algorithm. That is, the number of tuples satisfying $s_{i,k}$ in $\mathcal{A}^*$ is $f_{i,k}$. The same holds for every $i \in \overline{1,n}$, thus, $\mathcal{A}^*$ satisfies $\mathcal{Q}$.

Next, we show that CPS computes a representative sample. We do so by showing that for every Q_i in $\mathcal{Q}$, the probability CPS will return the answer A_i is the same as the probability that a representative non-optimal algorithm will return this answer, and vice versa.

Algorithm 2 Algorithm CPS

CPS$(\mathcal{Q}, C)$

Input: A dataset R and an MSSD query $(\mathcal{Q}, C)$
Output: A stratified sample $\mathcal{A}^*$

1: compute for $\mathcal{Q}$ a representative non-optimal answer $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ and find $F(A_i, \sigma), \forall \sigma \in [[\mathcal{Q}]], i \in \overline{1,n}$
2: formulate the IP according to Figure 3
3: using a solver, compute assignments to the decision variables of the IP
4: construct a new query Q' by creating a new stratum constraint $s(\sigma)$ for each $\sigma \in [[\mathcal{Q}]]$
5: $A' \leftarrow$ SSDA(Q'), where SSDA is some SSD algorithm
6: **for** $i = 1$ to n **do**
7: $\quad A_i^* \leftarrow \emptyset$
8: **for each** σ in $[[\mathcal{Q}]]$ **do**
9: $\quad M \leftarrow$ all the tuples of A' that are associated with σ
10: $\quad$ *(* these are all the tuples that satisfy σ in A' *)*
11: $\quad$ **for each** $\tau \subseteq I(\sigma)$ **do**
12: $\quad\quad M' \leftarrow X_\tau(\sigma)$ tuples of M (arbitrarily chosen)
13: $\quad\quad M \leftarrow M \setminus M'$
14: $\quad\quad$ **for each** i in τ **do**
15: $\quad\quad\quad A_i^* \leftarrow A_i^* \cup M'$
16: **return** $\cup_{i=1}^n A_i^*$

We can partition all the answers to Q_i into classes such that two answers A_i^1 and A_i^2 are in the same class if and only if $F(A_i^1, \sigma) = F(A_i^2, \sigma)$ for all $\sigma \in [[\mathcal{Q}]]$. The *probability* of a class is the probability that an arbitrary answer to Q_i will be in this class, i.e., the size of the class divided by the total number of answers to Q_i.

We now consider the probability of selecting an answer A_i from all the answers of its class. First, note that the stratum selections $[[\mathcal{Q}]]$ partition R into mutually disjoint sets, i.e., for every $\sigma_i, \sigma_j \in [[\mathcal{Q}]]$ holds $\sigma_i(R) \cap \sigma_j(R) = \emptyset$. So, to choose A_i, for each $\sigma \in [[\mathcal{Q}]]$, $F(A_i, \sigma)$ tuples are drawn out of $\sigma(R)$ tuples. Thus, the probability of randomly selecting A_i from all the answer of the class is $\prod_{\sigma \in [[\mathcal{Q}]]} (F(A_i, \sigma)/\sigma(R))$.

Suppose there are $|C_i|$ elements in the class of A_i and there is a total of T_i answers to Q_i, then the probability of selecting the answer A_i by a representative algorithm is $\frac{1}{T_i} = \frac{1}{|C_i|} \frac{|C_i|}{T_i} = \prod_{\sigma \in [[\mathcal{Q}]]} (F(A_i, \sigma)/\sigma(R)) \frac{|C_i|}{T_i}$.

The probability of returning A_i by CPS is the probability that an answer in the same class as A_i will be computed in Line 1 (this probability is $\frac{|C_i|}{T_i}$), multiplied by the probability that A_i will be selected among all the other answers of the class. Since in Lines 5-15, $F(A_i, \sigma)$ tuples are uniformaly selected from $\sigma(R)$ tuples, for each $\sigma \in [[\mathcal{Q}]]$, the probability of this selection is $\prod_{\sigma \in [[\mathcal{Q}]]} (F(A_i, \sigma)/\sigma(R))$. Hence, the probability of returning A_i as the answer to Q_i is the same for the representative algorithm and CPS.

We now show that CPS is optimal. Note that for $\sigma \in [[\mathcal{Q}]]$ and $\tau = \{i_1, i_2, \ldots, i_{m'}\} \subseteq I(\sigma)$, the term $c_\tau \cdot X_\tau(\sigma)$ (the sum of which is to be minimized according to Figure 3) represents the cost of surveying the population indicated by $\mathcal{A}^*$. Hence, the objective of the integer program, is to find the minimal cost without violating the representativeness of the result. Given some algorithm ALG_2 that provides a rep-

resentative sample, the frequencies $F(A_i, \sigma)$ have the same distribution as the distribution of these frequencies when using the algorithm employed in Line 1 of Algorithm 2. Thus, we can assume, without loss of generality, that ALG_2 is used in Line 1 of Algorithm 2. Now, since the integer program minimizes the costs with respect to these frequencies, we get that $c(\mathcal{A}_i^2) \geq c(\mathcal{A}^*)$ for every run, where $\mathcal{A}_i^2$ is the answers to ALG_2 and $\mathcal{A}^*$ is the answers to CPS. This holds for any algorithm ALG_2, hence, CPS is optimal.

5.2.5 Algorithm MR-CPS

There are two issues that need to be considered when examining the efficiency of CPS. The first is the number of decision variables and constraints, according to the definitions in Figure 3. In determining the number of stratum selections in $[[\mathcal{Q}]]$, notice that for each $Q_i \in \mathcal{Q}$, we can either skip Q_i or choose a single stratum constraint out of m_i options. Hence, the total number of stratum selections is $\prod_{i=1}^{n}(m_i + 1)$. Furthermore, the index selections τ are subsets of n indexes, so there are $|2^{\overline{1,n}}| = 2^n$ selection options. Hence, generating decision variables and constraints according to Figure 3, requires $2^n \cdot \prod_{i=1}^{n}(m_i + 1)$ iterations. This makes even just the formalization of the problem impractical in terms of running time and memory requirements, for an MSSD of significant size. Thus, next we present Algorithm *MR-CPS*, which is a (nearly-optimal) version of CPS, yet it is both efficient and scalable.

5.2.5.1 Calculating $[[\mathcal{Q}]]^*$.

Generating all the stratum selections of $[[\mathcal{Q}]]$ is not always necessary. If for some stratum selection σ there are no tuples in $\mathcal{A}$ to satisfy it, then this σ is redundant. In other words, if $F(A_i, \sigma) = 0$ for all $i \in \overline{1,n}$, then σ is redundant. To demonstrate this, consider adding to the MSSD from Example 6 a single SSD with two stratum constraints: $s_{3,1} = (age < 20, 10)$ and $s_{3,2} = (age \geq 20, 10)$. The stratum selection $\sigma = \{s_{1,2}, s_{2,2}, s_{3,1}\}$ is satisfied by women, or girls, whose age is below 20 and their salary is above \$200000. Thus, it is likely that $F(A_i, \sigma) = 0$ for any $i \in \overline{1,n}$. From Figure 3, it is apparent that if $F(A_i, \sigma) = 0$ for some σ then all the decision variables on its left hand side must also be assigned a value of 0, hence, they can be dismissed. Therefore, a stratum selection σ is generated only if it is satisfied by at least one tuple of the initial answer $\mathcal{A}$.

In the improved version of CPS, instead of using $[[\mathcal{Q}]]$, we use a set of only relevant stratum selections, denoted by $[[\mathcal{Q}]]^*$. Next, we explain how this subset is generated. For a tuple $t \in R$, the *stratum selection* of t, denoted $\sigma(t)$, is a selection $\sigma \in [[\mathcal{Q}]]$ such that t satisfies σ and such that $|\sigma|$ is maximal. For a tuple $t \in R$, the value of $\sigma(t)$ is calculated by iterating over every $Q_i \in \mathcal{Q}$ and selecting a stratum constraint that t satisfies, from Q_i (if one exists). The stratum selection $\sigma(t)$ is then inserted into a trie data structure [9] called the *stratum selection trie (SST)*, as illustrated in Figure 5. We create an SST for every answer $A_i \in \mathcal{A}$ by iterating over each $A_i \in \mathcal{A}$ and generating a corresponding SST from its tuples. The depth of each SST is n and in each level there can be at most m^* nodes, where $m^* = 1 + \max_{i \in \overline{1,n}}(m_i + 1)$. As a result, inserting a stratum selection of a tuple into the SST has $O(n \cdot m^*)$ time complexity. This is also the time complexity of looking up a stratum selection in an SST. Hence, creating an SST for each SSD query has $O\left(n \cdot m^* \cdot \sum_{i=1}^{n} |A_i|\right)$ time complexity.

$$\text{map}(null, t) \rightarrow \text{list}\left((\sigma(t), 1)\right)$$
$$\text{reduce}(\sigma, list(i_1, \ldots, i_N)) \rightarrow \text{list}\left(\left(\sum_{j=1}^{N} i_j\right)\right)$$

Figure 4: Calculating the stratum selection limits.

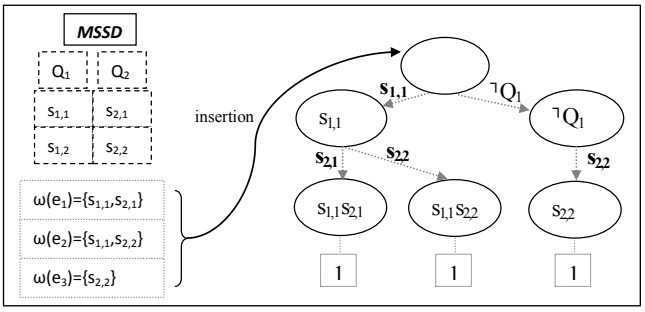

Figure 5: The SST of 2 SSDs and 3 tuples (and their corresponding stratum selections). Each leaf contains the instance count of a stratum selection.

By iterating over all the stratum selections in these SSTs we can derive $[[\mathcal{Q}]]^*$ which holds only the relevant stratum selections. Note that $|[[\mathcal{Q}]]^*| \leq min\left(|\cup_{i=1}^{n} A_i|, |[[\mathcal{Q}]]|\right)$, which means that $|[[\mathcal{Q}]]^*|$ is no larger than the sum of required sample sizes defined in the stratum constraints of each $Q_i \in Q$. As sample sizes tend to be relatively small, generating $[[\mathcal{Q}]]^*$ is feasible using the main memory of a single machine.

The leaf nodes of the SSTs contain an instance count for every inserted stratum selection. The SSTs can therefore be used to efficiently determine the values of $F(A_i, \sigma)$ for every $i \in \overline{1,n}$ and every $\sigma \in [[\mathcal{Q}]]^*$. Similarly, the limit of each stratum selection $L(\sigma)$ can be determined by creating an SST from R and checking the leaf nodes which contain the instance counts. For scalability, we use a simple MapReduce program, depicted in Figure 4 for obtaining these counts.

5.2.5.2 Linear Programming.

The second main issue that needs to be addressed to make CPS practical, is the integer programming, which is an NP-Hard problem. Instead of using an integer programming solver, we use a linear programming solver. Note that, a linear programming solver can assign non-integer values to decision variables. Given our semantics, this is problematic. The solution we use is to round the assignments to the lower value, i.e., $\lfloor X_\tau(\sigma) \rfloor$ for every decision variable.[4] As a result, after running a CPS based algorithm, the number of tuples may be slightly less than the required sample frequencies of the stratum constraints. We resolve this problem by duplicating the original SSDs and updating the required sample frequencies of every stratum constraint in each SSD to be equal to the residual frequency. That is, for every Q_i and for every $\sigma \in [[\mathcal{Q}]]$, if there are less than $F(A_i, \sigma)$ tuples in A_i, we randomly select the missing tuples from $\sigma(R)$ and add to A_i. Technically, we do so by running another phase of MSSD and adding to the answer the *residual answers*.

[4]Due to floating point quantization errors in the solver, we actually assign $\lfloor X_\tau(\sigma) + \epsilon \rfloor$ where $\epsilon = 0.0001$.

We need to show that the use of LP does not affect the representativeness of the sample. Recall the discussion in Section 5.2.4. For an answer A_i, the selection of the class is prior to the call to an LP or an IP solver, so it is not affected by the type of solver. As for the selection of $F(A_i, \sigma)$ tuples for σ, this is a uniform selection of $F(A_i, \sigma)$ tuples out of $\sigma(R)$, so it also has the same probability as in Section 5.2.4. Hence, the probability of an answer to be returned is as in the representative case.

Our experimental results in Section 6 show that in terms of survey cost, LP is almost as effective as IP.

5.2.5.3 *Running Time Analysis.*

For efficiency and scalability, MR-CPS employs MR-SQE and MR-MQE to answer SSD and MSSD queries. Calculating $L(\sigma)$ for every $\sigma \in \{\sigma(t)|t \in R\}$ to formulate the LP problem is achieved using the MapReduce program shown in Figure 4. It requires $O\left(|[[\mathcal{Q}]]^*| \cdot 2^n\right)$ iterations to create the decision variables and the constraints of the LP problem. Assuming the running time of the solver is L, then the formulation of the LP problem and its evaluation have $O\left(L \cdot n \cdot m^* \cdot 2^n\right)$ time complexity. This running time is exponential but only in the number of SSDs, which is expected to be small. Note that the running time of the LP component is independent of the data size. In terms of scalability, the only part of Algorithm MR-CPS that is unsuitable for parallel computation is the LP component. However, since the running time of the LP solver does not depend on the size of the data (only on the size of the query) the algorithm can be easily scaled to cope with larger datasets by adding machines to the distributed system.

6. EXPERIMENTAL EVALUATION

The goal of this section is to provide an experimental evaluation of the algorithms presented in Section 5. First, we illustrate the effectiveness of MR-CPS in terms of its ability to produce answers that reduce the expenses of surveys. We then analyze the efficiency and scalability of the algorithms by examining their running times.

6.1 Setting

In our tests we used the following dataset and queries.

6.1.1 *Dataset*

We used the DBLP[5] Bibliography dataset, which contains 1.7 million publications of more than one million authors. We extracted from it a list of computer science researchers (authors). Each author was assigned a set of attributes consuming 100 KB of storage. The SSD queries we issued only refer to the subset of attributes depicted in Table 1. (Note that the Dagum and Burr distributions are commonly used to model income.) There are obvious correlations between values of different columns, as in almost any realistic dataset. The total size of our dataset is slightly above 100 GB.

6.1.2 *MSSD Queries*

In order to effectively examine the different aspects of our algorithms we generated three groups of queries. To generate these groups we created a framework whose purpose is to efficiently generate strata. The strata are generated by partitioning the domains presented in Table 1 into subranges. The partition is into ranges of equal size, with an "error" of 10 percent, to create diversity. (Subranges are disjoint and their union covers the domain.) Every subrange is represented by a propositional formula, e.g., $fy \geq 1960$ *and* $fy \leq 1980$. The strata are created by randomly selecting attribute subranges so that every two strata are disjoint. Each stratum is defined as a disjunction of some subrange formulas.

Attr.	Domain	Description	Distribution
id	-	Author's unique id	-
name	-	Author's name	-
nop	[1, ..., 699]	Total number of papers	Dagum ($k = 0.68$, $\alpha = 0.52, \beta = 0.89, \gamma = 1$)
ayp	[0, ..., 40]	Average number of papers per year	Dagum ($k = 0.24$, $\alpha = 0.87, \beta = 0.66, \gamma = 1$)
myp	[0, ..., 140]	Maximum number of papers per year	Dagum ($k = 0.16$, $\alpha = 0.86, \beta = 0.78, \gamma = 1$)
fy	[1936, ..., 2013]	Year of first publication	Power Function ($\alpha = 7.75$, $a = 1936, b = 2013$)
ly	[1936, ..., 2013]	Year of last publication	Power Function ($\alpha = 11.83$, $a = 1936, b = 2013$)
cc	[1, ..., 1000]	Distinct coauthors for all papers	Burr ($k = 0.47$, $\alpha = 2.96, \beta = 3.05, \gamma = 0$)
ndcc	[1, ..., 2500]	Non distinct coauthors	Burr ($k = 0.32$, $\alpha = 2.92, \beta = 2.83, \gamma = 0$)
accpp	[0, ..., 129]	Average number of coauthors per paper	Dagum ($k = 0.98$, $\alpha = 3.41, \beta = 3.42, \gamma = 0$)

Table 1: Attributes of researchers in the dataset

We used this framework to generate three query groups, of different sizes. Let n denote the number of required SSDs, m_{sr} denote the number of subranges created for each attribute, and m_c be the number of subranges we combine (using disjunction) to define a single stratum constraint. Then, given these parameters, the total number of stratum constraints in an SSD is $m = (m_{sr})^{m_c}$. The three generated query groups were created using the following parameters.
Group *Small*: $n = 3, m_{sr} = 4, m_c = 2, m = 16$.
Group *Medium*: $n = 6, m_{sr} = 4, m_c = 3, m = 64$.
Group *Large*: $n = 9, m_{sr} = 4, m_c = 4, m = 256$.
For each query group, we created three copies of it—each copy is designed to create a sample on a different scale—creating samples of 100, 1000 and 10000 tuples. (The sample are, respectively, 0.01%, 0.1% and 1% of the data.)

In actual surveys, costs may vary according to the data collection method. To illustrate our approach we considered an interview cost of $4. This value is based on studies related to the optimal incentive necessary to produce survey participation [14]. To ease the creation of cost tuple coefficients, to indicate that sharing is not desired, we define a *penalty*, denoted $p_{\{i,j\}}$. A penalty is defined on SSD index selections of size 2 (i.e., each penalty refers to 2 SSDs). A penalty $p_{\{i,j\}}$ is added to every sharing cost c_τ for which $\{i, j\} \subseteq \tau$. Hence, using penalties it is easy to change cost tuples coefficients. When creating the cost tuples we initially set the cost of every two shared interviews to be the cost of a single interview (i.e., $4), and then we added a penalty of $10 to randomly chosen pairs of SSDs. Penalties were set to be $10 so that a penalty will cost more than two interviews and undesired sharing will not pay off.

6.1.3 *Environment*

The algorithms were implemented in Java over the Hadoop MapReduce framework. We created 11 virtual machines on Amazon EC2, each is a 64bit M1-Small class with a Linux Ubuntu 12.0.4 OS, 1.7GB of RAM and 160GB of storage. One of the VMs was designated as a master node running the Name Node and the Job Tracker, and the other 10 serve as

[5] `http://www.informatik.uni-trier.de/~ley/db/`

Dataset	Small	Medium	Large
cost CPS/cost MQE	62%	51%	47%

Table 2: Survey cost when using MR-CPS as the percentage of the survey cost when using MR-MQE.

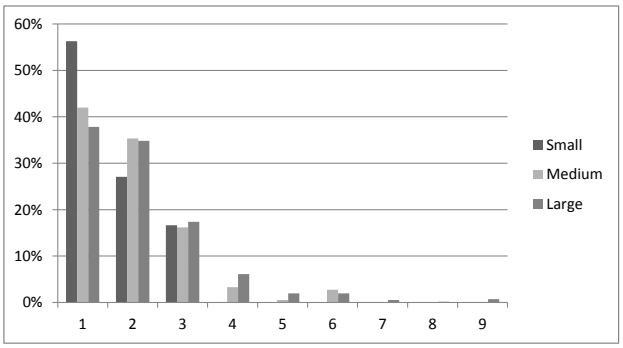

Figure 6: For $1 \leq i \leq 9$ (1=no sharing), the percentage of individuals assigned to i surveys by MR-CPS.

slave nodes, each running a Data Node and a Task Tracker. We used the Apache Math Commons[6] implementation of the Simplex algorithm to solve the LP problem in MR-CPS. We chose to use an open-source solver, however, using a more efficient solver can decrease the time spent on solving the LP problem and by this strengthen our method.

6.2 Results

In the experiments we tested the algorithms over the query groups *Small*, *Medium* and *Large*, producing samples of 100, 1000 and 10000 tuples.

6.2.1 Effectiveness

Effectiveness refers to the amount of sharing of individuals in different surveys and the saving that was achieved by sharing. It can be measured by comparing the number of individuals needed for all the surveys and the number of unique individuals that were actually selected. To test effectiveness, we measured the average costs of surveys produced by MR-MQE and MR-CPS, over 100 runs, as a function of the query-group size. We used MR-MEQ as a benchmark for MR-CPS, because it is oblivious to survey costs. Table 2 presents the percentage of survey costs saved by MR-CPS in comparison to MR-MQE. Note that the cost reduce is significant (exceeds 50%), and it grows as the size of the query group increases, because in a larger set of surveys there are more options to share individuals. To illustrate this, Figure 6 presents for $1 \leq i \leq 9$, the percentage of individuals assigned to i surveys by MR-CPS (average over 100 runs). This shows that MR-CPS assigns each individual to approximately 2 surveys, on average, whereas for comparison, in MR-MEQ individuals are selected independently, so the average sharing never exceeded 4%.

To test the influence of the distributions of values in the dataset, we created a synthetic dataset with the same set of users as those in DBLP and the same attributes as in Table 1, except that in this synthetic database, all the values were randomly chosen according to a uniform distribution (without any dependencies between the different attributes). We ran the tests on this synthetic dataset and compared the survey costs produced by MR-MQE and MR-CPS. The results are similar to those we report for the real dataset, because for some queries, the lack of correlations facilitates sharing while for other queries, the lack of correlations disrupts sharing. Overall, for a random set of queries, the distributions of values had no effect on the cost saving.

[6]http://commons.apache.org/math/

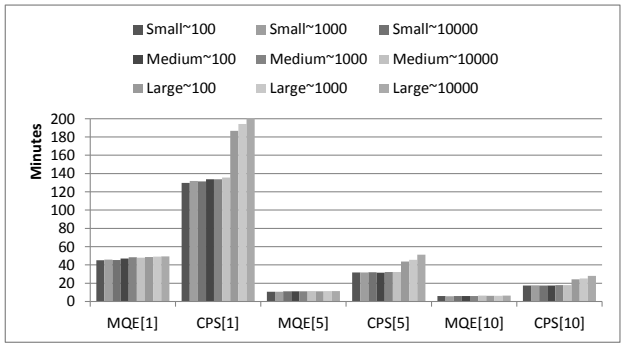

Figure 7: Running times, for the different query groups, on a cluster configuration of 1, 5 and 10 slaves (the number of slaves appears in square brackets). Results are averaged over 10 runs.

6.2.2 Optimality Analysis

Instead of using Integer Programming, MR-CPS uses Linear Programming to compute an optimal selection of individuals for the surveys. The LP solution is optimal, but it assigns non-integers numbers of individuals to surveys. Dealing with these non-integer assignments causes the solution to be non optimal. In Section 5.2.5.2 we refer to the assignments of the non-integer parts as residual answers. So, if C_{LP}, C_{IP} and C_{A} are the costs of the optimal LP solution, optimal IP solution and the answer of MR-CPS, respectively, then $C_{\text{LP}} \leq C_{\text{IP}} \leq C_{\text{A}}$. Thus, $C_{\text{A}} - C_{\text{IP}} \leq C_{\text{A}} - C_{\text{LP}}$, where $C_{\text{A}} - C_{\text{IP}}$ tells how far are the answers produced by MR-CPS from the optimum computed by the IP. To examine this difference, we collected statistics on the number of individuals contained in the residual answers. In our tests, the residual answers were at most 5.5% of the answers produced by MR-CPS. Hence, the difference between the cost of the computed answer and the cost of the optimal LP answer is approximately $0.055C_{\text{A}}$. Thus, in our experiments, $C_{\text{A}} - C_{\text{IP}} \leq 0.055C_{\text{A}}$. That is, the provided answer costs at most 5.5% more than of the optimal answer.

6.2.3 Efficiency

We measured the running times of the algorithms, to evaluate their efficiency and scalability. Figure 7 presents the running times of the algorithms on our cluster, for the three query groups, using cluster configurations of 1, 5 and 10 slaves. The results show that there is almost a linear improvement in the running times, for both MR-MQE and MR-CPS, when we add slave nodes. To investigate this, we measured the amount of time spent in each of the MapReduce phases. On average, in both algorithms around 70%, 28% and 1% of the running time are spent on the Mapper,

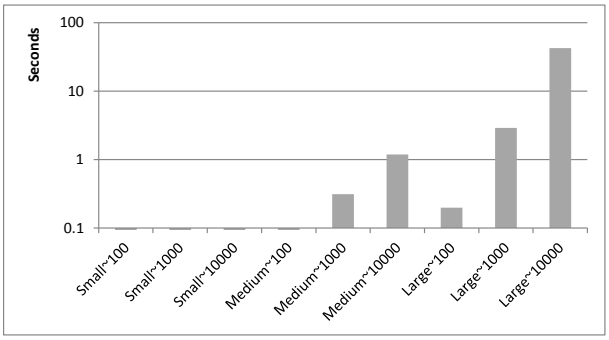

Figure 8: The average running times, in seconds, for formulating and solving the LP (log scale).

Combiner and Reducer phases, respectively. (The rest, no more than 1%, is spent on the LP.) So, around 98% of the work is done by the slave nodes and utilizes the scalability of Hadoop. Hence, the size of the data has a linear effect on the running time. By running the tests on our 100 GB dataset and on subsets of it, of size 50 GB and 10 GB, we confirmed the almost linear increase in running time.

Next, we examine the running time of MR-CPS. Recall that most of the running time of MR-CPS is spent on applying MR-SQE and MR-MQE (three times) and on formulating and solving the linear programming problem (LP). The LP solver is the only component whose running time cannot be improved merely by adding nodes, thus, it is important to understand its effect on MR-CPS. Figure 8 depicts the average running times devoted to solving the LP. Note that in all cases the running time of the LP solver is in the order of seconds. Thus, it is insignificant in comparison to the total running time of MR-CPS. Figures 7 and 8 show that the running times of MR-CPS are about 3 times longer than the running times of MR-MQE, and by this they confirm that the LP solver has almost no effect on the running times. This also shows that one node is enough for solving the LP.

We now discuss the effect of the query-group size on the running time of MR-CPS (see Figure 7). Recall that in Line 5 of Algorithm 2, MR-SQE is issued on a query whose size is proportional to $|[[\mathcal{Q}]]^*|$. This size is bounded by the number of stratum selections whose frequency is non-zero, but it may still be exponential in the number of queries. This causes a noticeable slowdown, for the Large query group. However, since the computation time of MR-SQE is linear in the number of slave nodes, this effect can be alleviated by adding more nodes (as shown in Figure 7).

7. CONCLUSION

We studied the problem of applying stratified sampling for selecting samples of a population from large-scale, distributed social networks. There are various costs associated with conducting surveys based on samples, such as data acquisition costs, anonymization of user data, verification of user authenticity, interview costs, coping with survey fatigue, etc. In this paper, we show that by conducting multiple surveys in parallel, a significant saving of expenses can be made. Based on the dependencies between surveys, sharing individuals to reduce survey expenses may be desired in some cases, and should be avoided in other cases. We present a framework for sharing individuals in different surveys and consider the goal of finding an answer that minimizes the survey expenses. We show that this problem is NP-Hard, and we provide a distributed heuristic algorithm for it, namely MR-CPS. (MR-CPS is a heuristic because it uses Linear Programming instead of Integer Linear Programing). Our experimental evaluation shows that CPS can significantly reduce the overall costs of conducting multiple surveys in parallel. We demonstrate the efficiency of MR-CPS by showing that it can process multiple stratified sampling queries over a dataset containing more than 100 GB of data, in an order of a few minutes, on a cluster containing low-end virtual machines; and we show that scalability can be easily achieved by adding nodes to the cluster.

8. ACKNOWLEDGMENTS

This research was supported in part by the Israel Science Foundation (Grant 1467/13) and by the Isreali Ministry of Science and Technology (Grant 3-9617).

9. REFERENCES

[1] A. Chaudhuri and H. Stenger. *Survey Sampling Theory and Methods*. Taylor and Francis Group, LLC, 2005.

[2] CNET. `http://news.cnet.com/8301-1023_3-57484991-93/facebook-8.7-percent-are-fake-users/`, 2012.

[3] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Optimal sampling from distributed streams. In *PODS*, pages 77–86, New York, NY, USA, 2010. ACM.

[4] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, Jan. 2008.

[5] W. A. Fuller. *Sampling Statistics*. Wiley Series in Survey Methodology, 2009.

[6] M. Gjoka, C. Butts, M. Kurant, and A. Markopoulou. Multigraph sampling of online social networks. *Selected Areas in Communications, IEEE Journal on*, 29(9):1893–1905, 2011.

[7] R. Grover and M. J. Carey. Extending map-reduce for efficient predicate-based sampling. In *ICDE '12*, pages 486–497, Washington, DC, USA, 2012.

[8] G. Kalton. *Intro to Survey Sampling*. Sage, 1983.

[9] D. Knuth. *The Art of Computer Programming: Sorting and Searching*. Addison-Wesley, 1997.

[10] M. Kurant, M. Gjoka, C. T. Butts, and A. Markopoulou. Walking on a graph with a magnifying glass: stratified sampling via weighted random walks. In *SIGMETRICS '11*, pages 281–292. ACM, 2011.

[11] M. Kurant, A. Markopoulou, and P. Thiran. On the bias of BFS (Breadth First Search). In *International Teletraffic Congress (ITC)*, pages 1–8, 2010.

[12] N. Laptev, K. Zeng, and C. Zaniolo. Early accurate results for advanced analytics on mapreduce. *Proc. VLDB Endow.*, 5(10):1028–1039, June 2012.

[13] F. Olken and D. Rotem. Random sampling from database files: a survey. In *SSDBM*, pages 92–111, Charlotte, NC, USA, 1990.

[14] J. H. Schuh. *Assessment Methods for Student Affairs*. John Wiley and Sons, 2011.

[15] S. Tirthapura and D. P. Woodruff. Optimal random sampling from distributed streams revisited. In *Proceedings of the 25th international conference on Distributed computing*, DISC'11, pages 283–297. Springer-Verlag, 2011.

[16] J. S. Vitter. Random sampling with a reservoir. *ACM TOMS*, 11:37–57, March 1985.

[17] M. Vojnovic, F. Xu, and J. Zhou. Sampling based range partition methods for big data analytics. Technical report, Microsoft Research, 2012.