Estimating the Number and Sizes of Fuzzy-Duplicate Clusters

Arvid Heise Gjergji Kasneci Felix Naumann
Hasso-Plattner Institute (HPI)
Postdam, Germany
first.last@hpi.de

ABSTRACT

Duplicates in a dataset are multiple representations of the same real-world entity and constitute a major data quality problem. This paper investigates the problem of *estimating* the number and sizes of duplicate record clusters in advance and describes a sampling-based method for solving this problem. In extensive experiments, on multiple datasets, we show that the proposed method reliably estimates the number of duplicate clusters, while being highly efficient.

Our method can be used a) to measure the dirtiness of a dataset, b) to assess the quality of duplicate detection configurations, such as similarity measures, and c) to gather approximate statistics about the true number of entities represented in the dataset.

Categories and Subject Descriptors

H.2.0 [**Database Management**]: General

General Terms

Algorithms, Theory

Keywords

Data integration, estimation, duplicate, cluster, pair

1. DUPLICATE DETECTION

Duplicates in a dataset are multiple representations of the same real-world entity and constitute a major data quality problem. The consequences range from unsatisfied employees and customers, to incorrect analyses that lead to poor business decisions.

Duplicate detection (aka. record linkage, entity resolution, etc.) is the task of finding duplicate records, so that they can be subsequently eliminated or merged. Various techniques have been proposed to solve the two main challenges: When to declare a pair or cluster of records as a duplicate and how to efficiently find them all without naïvely searching the complete Cartesian product of possible record pairs. While the first challenge is usually solved by employing specific *similarity measures* and thresholds, the latter is solved by employing smart *candidate selection* techniques, such as the Sorted Neighborhood (SNM) [7] or (non-)overlapping Blocking [11].

However, all methods have in common that different parameters (e.g., thresholds, window sizes, and blocking keys) influence the result in different, often non-obvious, ways. Usually, these parameters are iteratively tweaked in successive runs and the respective outcome evaluated against a validation set or by close examination. This process consumes much time and resources and can be frustrating.

In practical scenarios, prior to the full and costly detection of all duplicates, the user might be rather interested to quickly gain a vague idea on the approximate sizes of duplicate clusters in the dataset, based on a user-defined similarity measure. If done in a simplistic way, the estimation of the cluster sizes may be as expensive as the retrieval of the duplicates itself. Hence, to be practically relevant, any method for estimating the numbers and sizes of duplicate clusters in a dataset would have to run much faster than any duplicate detection algorithm on the same dataset. In addition to being a means for estimating the degree of dirtiness of a dataset, such a method could be used for fine-grained analyses of the distribution of duplicates and thus help estimating the costs of various duplicate detection strategies.

In this paper, we propose a reliable and highly efficient technique to estimate the sizes of duplicate clusters in a dataset. The overarching vision of this work is to provide useful tools for interactive data cleansing systems, and thus provide users with a means for quickly calculating approximate statistics, planning resources accordingly (e.g., for the query optimizer), and analyzing data cleansing strategies.

1.1 Problem statement

In the following, we informally describe the challenges and formally define the key concepts.

Definition 1 (Duplicate Clusters). *The output of a duplicate detection run on a dataset $\mathcal{R}$ is the set of duplicate clusters $\mathcal{C}_\mathcal{R}$, where each $C \in \mathcal{C}_\mathcal{R}$ represents the set of all representations of the same real-world entity according to the used duplicate detection algorithm. Formally, $\mathcal{C}_\mathcal{R}$ is a partition over the equivalence relation $\sim$, i.e., the quotient set $\mathcal{R}/\sim := \{C \subseteq \mathcal{R} \mid \forall r_i, r_j \in C : r_i \sim r_j\} = \mathcal{C}_\mathcal{R}$.*

We are especially interested in the size distribution of the duplicate clusters.

CIKM'14, November 3–7, 2014, Shanghai, China.
Copyright 2014 ACM 978-1-4503-2598-1/14/11 ...$15.00.
http://dx.doi.org/10.1145/2661829.2661885.

Definition 2 (Cluster Size Histogram). *The cluster size histogram $\mathcal{H}_\mathcal{R}$ is the absolute histogram over the number of clusters of a certain size from the set of all clusters $\mathcal{C}_\mathcal{R}$. We denote the i-th entry in the histogram with $\mathcal{H}^i_\mathcal{R} = |\{C \in \mathcal{C}_\mathcal{R} : |C| = i\}|$ corresponding to the number of clusters of size i.*

The duplicate clusters and our estimations highly depend on the parameters of a **duplicate detection run**. In this paper, we assume the traditional, three-phase duplicate detection approach, which starts with a **candidate selection** method, such as SNM or Blocking, to retrieve a list of record pairs that have a high probability to be duplicates. It then performs the **candidate comparison**, which may apply similarity measures and fixed thresholds on the candidates. Finally, a **clustering** algorithm (e.g., transitive closure) generates duplicate clusters from the given set of pairs. Most other duplicate detection approaches can be simulated with this approach, so that our estimation method is transferable to other approaches as well.

All algorithms and parameters influence the result of a duplicate detection run. Therefore, our estimations need to be considered in the context of the actual parameters and we accordingly define the problem we aim to solve:

Given a configured duplicate detection run on a dataset $\mathcal{R}$, Duplicity Estimation *determines the estimated cluster size histogram $\mathcal{H}_\mathcal{R}{}' \approx \mathcal{H}_\mathcal{R}$ with much less effort than the run itself would take.*

1.2 Use cases and contributions

We envision four types of applications of our presented techniques.

Data Quality Estimation. In data integration processes, experts have to decide which data sources to integrate. Usually, they need to trade effort with the expected data quality gain. Our methods help with both: We can estimate the duplicity of one data source as an indicator for its quality and we can estimate the overlap between two data sources.

Interactive Systems. Tweaking the parameters of a duplicate detection task can be cumbersome, especially if it takes a long time to calculate the results. With our techniques, users can quickly see the approximate numbers of duplicates and some examples.

Approximate Statistics. For complex statistical analyses to reach business decisions, additional data sources are often integrated in an ad-hoc fashion. Since these statistics usually aggregate data, a full duplicate detection unnecessarily delays the results. With our techniques, we can efficiently answer questions, such as 'How many distinct customers do two companies have after a potential fusion?'.

Query Optimization. Recent development in (scientific) data workflows aims to optimize a large variety of operators. A complete integration of these operators into the optimizer requires an estimation of the output cardinality and reordering rules. In this paper, we address the former.

We address the duplicity estimation problem with the following contributions:

(1) We analyze the effects of sampling onto the cluster size histogram in Section 2.

(2) In Section 3, we develop a random walk model based on these insights to iteratively estimate the histogram.

(3) Section 4 introduces a bootstrapping method to speed up the convergence.

(4) We extensively evaluate our method on three datasets with different characteristics in Section 5.

(5) Section 6 extends the basic model to incorporate candidate selection and clustering.

Sections 7 and 8 close with a discussion of related work and a conclusion.

2. EFFECTS OF RANDOM SAMPLING ON DUPLICATES

To extrapolate findings on a sample $\mathcal{S} \subseteq \mathcal{R}$ to the original dataset $\mathcal{R}$, it is important to understand how sampling changes the number of duplicates and the cluster size histogram. In this section, we develop a probabilistic model of the sampling process.

We denote with $n = |\mathcal{S}|$ the number of samples, $N = |\mathcal{R}|$ the size of the original dataset, and with r or r_i a single record in $\mathcal{R}$, and consider random sampling with a sample rate $p(r \in \mathcal{S}) = \frac{n}{N}$. In the following, we first examine how the number of duplicate pairs changes for a random sample and then quantify the expected cluster size histogram.

2.1 Duplicate pairs

In some cases, it is already enough to know the number of duplicate pairs instead of the complete cluster size histogram. For example, when integrating two clean data sources (i.e., without duplicates within each source), we would match only records from different sources and would split clusters having a size larger than two.

We define duplicates as a subset of the *pair set*, such that each pair satisfies the equivalence relation $\sim$.

Definition 3 (Pair set). *A pair set of $\mathcal{R}$ contains all pairs of different records of $\mathcal{R}$ once:*
$\mathcal{R}^{<2} = \{(r_i, r_j) \mid (r_i, r_j) \in \mathcal{R} \times \mathcal{R}; i, j \in \{1, \ldots, N\} \wedge i < j\}$.

Definition 4 (Duplicate pairs). *The set of the duplicate pairs $\mathcal{D}_\mathcal{R}$ contains all pairs of duplicates in $\mathcal{R}$:*
$\mathcal{D}_\mathcal{R} = \{(r_i, r_j) \mid (r_i, r_j) \in \mathcal{R}^{<2} \wedge r_i \sim r_j\}$.

We now inspect the ratio of duplicate pairs in the sample $\mathcal{S}$ with respect to the original dataset $\mathcal{R}$. First, we estimate how many duplicate pairs $\mathcal{D}_\mathcal{S}$ can be found in the sample. The expected number of duplicate pairs $E[|\mathcal{D}_\mathcal{S}|]$ depends on the probability $p(d \in \mathcal{D}_\mathcal{S})$ that we sample a duplicate pair $d \in \mathcal{D}_\mathcal{R}$ in the sample $\mathcal{S}$.

$$\begin{aligned} E[|\mathcal{D}_\mathcal{S}|] &= |\mathcal{R}^{<2}|\ p(d \in \mathcal{D}_\mathcal{S}) \\ &= |\mathcal{R}^{<2}|\ p(d \in \mathcal{D}_\mathcal{R})\ p(d \in \mathcal{D}_\mathcal{S} \mid d \in \mathcal{D}_\mathcal{R}) \end{aligned} \tag{1}$$

Because random sampling is independent of the probability that a pair is a duplicate, the second probability is equal to the probability that we draw a specific pair during sampling $p(d \in \mathcal{S}^{<2})$. We can further factor in that we have $\binom{|\mathcal{X}|}{2}$ possibilities to draw a pair from $\mathcal{X}^{<2}$.

$$\begin{aligned} E[|\mathcal{D}_\mathcal{S}|] &= |\mathcal{R}^{<2}|\ p(d \in \mathcal{D}_\mathcal{R})\ p(d \in \mathcal{S}^{<2}) \\ &= |\mathcal{D}_\mathcal{R}|\ p(d \in \mathcal{S}^{<2}) = |\mathcal{D}_\mathcal{R}|\ \frac{\binom{n}{2}}{\binom{N}{2}} \\ &= |\mathcal{D}_\mathcal{R}|\ \frac{n\ (n-1)}{N\ (N-1)} \end{aligned} \tag{2}$$

Hence, the number of duplicate pairs increases quadratically with the sample size.

Example. Consider a dataset with 1000 tuples and 500 duplicate pairs. If we sampled 100 records, we would expect $|\mathcal{D}_\mathcal{S}| = 500\frac{100\cdot 99}{1000\cdot 999} \approx 4.95$ duplicate pairs in the sample. For a 200 records sample, the expected value would be ≈ 19.92.

Apparently, *extrapolating* from such a sample to estimate the original (i.e., actual) number of duplicates is highly sensitive to the sample variance. In the above example, we should receive five duplicates in most samples to obtain a good estimate of $|\mathcal{D}_\mathcal{R}| = 5\ \frac{1000\cdot 999}{100\cdot 99} \approx 504.55$. However, even a slight variation of one sampled duplicate changes the result by approximately 100 duplicate pairs.

2.2 Duplicate clusters

We now generalize the model to arbitrary cluster sizes. Similar to estimating the number of duplicate pairs, we use the number of permutations that can occur during sampling.

Our problem is related to the multivariate hypergeometric distribution; that is, drawing a sample without replacement from a multi-type population. We can map our problem to drawing a number of colored balls without replacement from an urn: Each different duplicate cluster of size i is represented by i balls of the same color. However, we are not interested in how many balls of a specific color are drawn but only in how many balls of the same color would be drawn on average if the random sampling experiment was repeated infinitely many times.

We can thus model our expected value by using the indicator function $[\![|C| = i]\!]$ that is 1 iff the size of a duplicate cluster C is i and 0 otherwise. We iterate over all possible events that can yield a duplicate cluster of size i in a sample of size n. Each outcome is weighted with $\binom{N}{n}^{-1}$, i.e., the reciprocal to the number of possibilities of randomly sampling n out of N elements without replacement:

$$E[\mathcal{H}^i_\mathcal{S}] = \frac{1}{\binom{N}{n}} \sum_{C\in\mathcal{C}_\mathcal{R}} \left(\sum_{\mathcal{S}\subseteq\mathcal{R}\wedge|\mathcal{S}|=n} [\![|C\cap\mathcal{S}| = i]\!] \right) \tag{3}$$

The first sum iterates over all clusters in $\mathcal{R}$ and the second sum enumerates all possible sample outcomes w.r.t. a specific cluster. Now, we can replace the second sum with the probability that we draw i elements from a cluster in the original dataset, which can be expressed as:

$$E[\mathcal{H}^i_\mathcal{S}] = \frac{1}{\binom{N}{n}} \sum_{C\in\mathcal{C}_\mathcal{R}} \binom{|C|}{i}\binom{|\mathcal{R}\setminus C|}{n-i} \tag{4}$$

Intuitively, there are $\binom{N}{n}$ possible sample permutations. For a cluster of size i in the sample, we draw i records from the cluster C, add $n-i$ records from the remaining dataset $\mathcal{R}\setminus C$ to fill up the sample, and normalize the result over all sample permutations.

Because the probabilities of drawing i elements from two different clusters with the same sizes are the same, we can simplify the calculation by collapsing the cases. For each cluster size k in the original dataset, we calculate the probability once and multiply it by the number of clusters of the given size $\mathcal{H}^k_\mathcal{R}$.

$$E[\mathcal{H}^i_\mathcal{S}] = \sum_{k=i}^{|R|} \mathcal{H}^k_\mathcal{R} \frac{\binom{k}{i}\binom{N-k}{n-i}}{\binom{N}{n}} \tag{5}$$

Ex.	Original histogram $\mathcal{H}^1_\mathcal{R}$	$\mathcal{H}^2_\mathcal{R}$	$\mathcal{H}^3_\mathcal{R}$	$\mathcal{H}^4_\mathcal{R}$	Exp. histogram from sampling $\mathcal{H}^1_\mathcal{S}$	$\mathcal{H}^2_\mathcal{S}$	$\mathcal{H}^3_\mathcal{S}$	$\mathcal{H}^4_\mathcal{S}$
1	1000	0	0	0	100.000	0.000	0.000	0.000
2	0	500	0	0	90.090	4.955	0.000	0.000
3	0	0	0	250	73.095	12.088	0.878	0.023
4	600	100	40	20	93.604	3.030	0.109	0.002

Table 1: Four different cluster size histograms for a dataset of size 1000 (left) and the expected histograms from sampling 100 records (right).

For ease of presentation, we assume $\binom{a}{b} = 0$ for $a < b$ in this paper. Alternatively, ranges in sum formulas must be adjusted to guarantee $a \geq b$. Now we are interested in understanding which clusters the sampled elements come from. Hence, we define a random variable $X^i_{S,n}$ representing whether an element from a cluster of size i occurs in a sample S of size n. The probability of this event happening is:

$$p(X^i_{S,n} = 1) = \frac{i}{n} \sum_{k=i}^{|R|} \mathcal{H}^k_\mathcal{R} \frac{\binom{k}{i}\binom{N-k}{n-i}}{\binom{N}{n}} \tag{6}$$

As discussed earlier, the sum in the above formula represents all possibilities to generate a cluster of size i in a sample of size n. Given that we have such a sample of size n at hand (in which there occurs a cluster of size i), the probability of picking an element from an i-cluster is i/n.

Table 1 shows four different cluster size histograms of a dataset with 1000 records and corresponding expected histograms for a sample of 100 records calculated with Equation 5. The first row constitutes a sanity check: If all 1000 records are duplicate-free, all 100 samples also must be duplicate-free. The second row corresponds to the previous example of 500 duplicate pairs in the previous section and indeed returns the same result. Interestingly, in the third row, even though the dataset consists of 250 clusters with size four, sampling one complete cluster of size four is highly unlikely with a sample size of 100 (one in 43 sample runs). Lastly, the expected sampling histogram of a somewhat realistically distributed dataset in the fourth row shows that sampling clusters of size 3 and 4 is even more unlikely.

3. EXTRAPOLATING FROM SAMPLE RESULTS

In the last section, we deduced Equation 5 to calculate the expected histogram from a given histogram. The calculation may be interpreted as multiplication of a matrix with transition probabilities $T_\mathcal{S}$ and the cluster size histogram interpreted as a vector $\vec{\mathcal{H}_\mathcal{R}}$.

$$\vec{\mathcal{H}_\mathcal{S}} = T_\mathcal{S}\vec{\mathcal{H}_\mathcal{R}} \tag{7}$$

$$T_\mathcal{S} = \begin{pmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,k} \\ 0 & s_{2,2} & \cdots & s_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_{i,k} \end{pmatrix} \tag{8}$$

$$s_{i,k} = \frac{\binom{k}{i}\binom{N-k}{n-i}}{\binom{N}{n}} \tag{9}$$

$$\vec{\mathcal{H}_\mathcal{R}} = (\mathcal{H}^1_\mathcal{R}\mathcal{H}^2_\mathcal{R}\cdots\mathcal{H}^i_\mathcal{R})^T \tag{10}$$

We could now treat the matrix multiplication as a system of linear equations, which can be exactly solved if the sample size is equal to or greater than the maximum cluster in $\mathcal{R}$, i.e., $|\mathcal{S}| \geq max_i(\mathcal{H}^i_{\mathcal{R}} > 0)$, and we would sample the expected numbers $E[\mathcal{H}_{\mathcal{S}}]$.

However, as seen in the last examples of Table 1, it becomes increasingly improbable to sample a larger cluster with random sample resulting in two drawbacks of the exact approach. First, we could sample a large cluster despite the low odds and would thus heavily overestimate the number of clusters in the original dataset. In fact, we would even receive an impossible high number of clusters with the exact method that needs to be compensated with negative histogram counts of smaller clusters. Second and more probable, we would not sample a large cluster at all and thus underestimate the number of clusters.

3.1 Probabilistic solution with random walk

We present a solution based on a random walk approach to probabilistically estimate from which original cluster a sample cluster has been drawn. The main intuition is to maximize the knowledge that we can obtain from a certain random sample and then successively enlarge the sample until we are confident with the estimation.

To extrapolate the sample histogram $\mathcal{H}_{\mathcal{S}}$ to the estimation $\mathcal{H}_{\mathcal{R}}$, we use the matrix $T_{\mathcal{R}}$, whose components are the conditional probabilities $p(X^k_R = 1 \mid X^i_{S,n} = 1)$, where $X^k_R = 1$ is a random variable representing whether a cluster of size k is present in $\mathcal{R}$.

$$\vec{\mathcal{H}_{\mathcal{R}}} = T_{\mathcal{R}} \vec{\mathcal{H}_{\mathcal{S}}} \tag{11}$$

$$T_{\mathcal{R}} = \begin{pmatrix} p_{1,1} & 0 & \cdots & 0 \\ p_{2,1} & p_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ p_{k,1} & p_{k,2} & \cdots & p_{k,i} \end{pmatrix} \tag{12}$$

$$\begin{aligned} p_{k,i} &= p(X^k_R = 1 \mid X^i_{S,n} = 1) \\ &= \frac{p(X^i_{S,n} = 1 \mid X^k_R = 1) p(X^k_R = 1)}{\sum_{j=i} p(X^i_{S,n} = 1 \mid X^j_R = 1) p(X^j_R = 1)} \end{aligned} \tag{13}$$

The last equation shows the two pieces of information that we need to calculate $T_{\mathcal{R}}$. The first term is the conditional probability $p(X^k_R = 1 \mid X^i_{S,n} = 1)$ that correspond to the mirrored entry of $T_{\mathcal{S}}$, which in turn depend only on $|R|$ and $|S|$. The second term $p(X^k_R = 1)$ corresponds to an entry in $\vec{\mathcal{H}_{\mathcal{R}}}$, which we actually want to estimate. In the denominator, we normalize over all cluster sizes in the sample.

Because we need prior knowledge of $\vec{\mathcal{H}_{\mathcal{R}}}$ to calculate $T_{\mathcal{R}}$, the iterative calculation of $\vec{\mathcal{H}_{\mathcal{R}}}$ is an obvious choice. We combine the transition matrices from Equations 7 and 11 to a matrix M_R:

$$\vec{\mathcal{H}_{\mathcal{R}}} = T_{\mathcal{R}} T_{\mathcal{S}} \vec{\mathcal{H}_{\mathcal{R}}} = M_R \vec{\mathcal{H}_{\mathcal{R}}} \tag{14}$$

We clearly see that $\vec{\mathcal{H}_{\mathcal{R}}}$ is the principal eigenvector of M_R. Therefore, we propose a random walk approach to start with a rough initial estimation $\mathcal{H}_{\mathcal{R}(0)}$ and iteratively refine the estimation until the convergence criterion is met.

Algorithm 1 consists of two nested convergence loops. In the outer loop (Lines 2–12), the algorithm first enlarges the sample and calculates the relative histogram $\mathcal{H}_{\mathcal{S}}$ by performing naïve duplicate detection on the sample. In Section 6, we show how advanced candidate selection can be used instead. It also calculates the sampling transition matrix $T_{\mathcal{S}}$[1], which depends on the current size of the sample and $|R|$.

The inner loop (Lines 7–11) performs the random walk to find a suitable $\mathcal{H}_{\mathcal{R}}$ for the current $\mathcal{H}_{\mathcal{S}}$. In each iteration, the algorithm first calculates the current transition matrix $T_{\mathcal{R}(i)}$ with the previous estimate $\vec{\mathcal{H}_{\mathcal{R}(i)}}$. This transition matrix is then used to calculate a better estimate $\vec{\mathcal{H}_{\mathcal{R}(i+1)}}$. The inner loop terminates when the estimate converges; that is, the L1-distance between two successive estimations is below a given threshold ε (i.e., less than ε clusters are changed), or 100 Power iterations [8] are reached, which avoids overfitting to early samples and speeds up the overall computation.

```
   Input  : Initial estimate $\vec{\mathcal{H}_{\mathcal{R}(0)}}$,
            convergence threshold $\varepsilon$,
            minimum number of samples $minS$
   Output: Estimated original histogram $\vec{\mathcal{H}_{\mathcal{R}}}$
1  $i, j \leftarrow 0$;
2  repeat
3      $j \leftarrow i$;
4      $\mathcal{S}_{(j)} \leftarrow$ sample;
5      calculate $\vec{\mathcal{H}_{\mathcal{S}(j)}}$ from $\mathcal{S}$ ;
6      calculate $T_{\mathcal{S}}$ for current sample size ;
       /* Power iterations to find $\vec{\mathcal{H}_{\mathcal{R}}} = T_{\mathcal{R}} \vec{\mathcal{H}_{\mathcal{S}}}$      */
7      repeat
8          calculate $T_{\mathcal{R}(i)}$ with $\vec{\mathcal{H}_{\mathcal{R}(i)}}$;
9          $\vec{\mathcal{H}_{\mathcal{R}(i+1)}} \leftarrow T_{\mathcal{R}(i)} \vec{\mathcal{H}_{\mathcal{S}(j)}}$;
10         $i \leftarrow i + 1$ ;
11     until $||\vec{\mathcal{H}_{\mathcal{R}(i)}} - \vec{\mathcal{H}_{\mathcal{R}(i-1)}}||_1 < \varepsilon$ or $i > j + 100$;
       /* Enlarge sample until glob. converged    */
12 until $||\vec{\mathcal{H}_{\mathcal{R}(j)}} - \vec{\mathcal{H}_{\mathcal{R}(i)}}||_1 < \varepsilon$ and $|S| \geq minS$;
```

Algorithm 1: Two-phase random walk.

Having an estimate for a certain, possibly small sample yields a high risk of overfitting to a poor sample. We thus rerun the random walk several times with the outer loop and enlarge the sample until 1) the estimates also converge across samples and 2) we sampled a given minimum number of records. The first criterion causes the algorithm to eventually converge to a well fitting estimate for the current sample and the second criterion avoids the algorithm being stuck in local maximum for poor initial samples.

3.2 Properties of the random walk

Algorithm 1 iteratively computes the principal eigenvector of the matrix M_R (see Equation 14). To show that this computation is well-defined, we have to show that real principal eigenvectors of the matrix exist.

We know that every quasi-positive, and thus every positive, matrix is ergodic and therefore has a principal eigenvector. For M_R, it is straight-forward to show that indeed all components have to be positive. All components of $T_{\mathcal{S}}$ and $T_{\mathcal{R}}$ are non-negative, so that no component in the product can be negative. Further, the first row of $T_{\mathcal{S}}$ and the first column of $T_{\mathcal{R}}$ are filled with positive numbers. Because the

[1]For $n > k$, we calculate the more efficient $s_{k,i} = \frac{\binom{n}{i}\binom{N-n}{k-i}}{\binom{N}{k}}$, which is derived through symmetry of binomial coefficients

matrix multiplication each component in M_R involves the multiplication of the first row of $T_{\mathcal{S}}$ and the first column of $T_{\mathcal{R}}$, we can conclude that each component must be positive. Intuitively, from any estimated cluster of size k, we could have sampled exactly one element, which could have been drawn from any other cluster size. Thus, we can change the estimation for one record in each iteration to a completely different cluster size.

Moreover for such matrices, the principal eigenvectors can be effectively approximated through Power iterations [8], which is represented also in the inner loop of the algorithm. These facts lay the mathematical foundation for the correctness of Algorithm 1.

THEOREM 1. *Algorithm 1 effectively approximates the principal eigenvector of the matrices M_R from Equation 14.*

Another interesting fact about Algorithm 1 follows from the Law of Large Numbers.

THEOREM 2. *The estimated variance of $\mathcal{H}_{\mathcal{R}}$ as computed by Algorithm 1 converges to the true variance of the distribution of cluster sizes in R.*

The estimation model reliably estimates the actual duplicate histogram even for poor initial estimates $\vec{\mathcal{H}}_{\mathcal{R}(0)}$ as we experimentally show in Section 5. Nevertheless, we also show that better initial estimates lead to faster convergence. Thus, in the next section we provide a quick method for providing a good $\vec{\mathcal{H}}_{\mathcal{R}(0)}$.

4. FOCUSED SAMPLING OF LARGER CLUSTERS

In this section, we propose a bootstrapping method to quickly sample larger clusters and retrieve an initial estimate $\mathcal{H}_{\mathcal{R}(0)}$ that covers a broad spectrum of cluster sizes. Random sampling is not adequate for quickly generating viable estimates of the proportions of larger clusters ($|C| > 2$, see Section 2). Therefore, our focused sampling method specifically *oversamples* larger clusters and then counter-balances it.

Up to this point, we did not assume anything about the similarity measure used in the candidate comparison. For the following focused sample approach we have to assume monotonicity.

DEFINITION 5 (MONOTONICITY). *A similarity measure is* monotonic *if the overall similarity does not decrease if the similarity of an attribute value increases.*

The assumption is quite intuitive and should hold for most similarity measures. Counterexamples occur mostly for similarity measures that are based on non-linear weighted combinations of features or mixtures of negatively and positively weighted features. We use this assumption to sample duplicates in a focused way: We specifically draw from groups of samples with the *same value in at least one attribute.* For any similarity measure *sim*, the similarity between equal values is maximal and thus by monotonicity we know that for a given attribute $A \in \mathcal{A}$:

$$p(r_1 \sim r_2 \mid \pi_A(r_1) = \pi_A(r_2)) \geq p(r_1 \sim r_2) \qquad (15)$$

The equations holds for attributes A that are used in the similarity measure. Nevertheless, intuitively the same

```
   Input  : Sample size $s$, randomness ratio $\rho$
   Output: Initial estimate $\mathcal{H}_{\mathcal{R}(0)}$
 1 $\mathcal{S} \leftarrow$ random sample of $s \cdot \rho$ records;
 2 Choose suitable attributes $\mathcal{A}$ in dataset (Equation 16);
 3 Initialize focused sample $\mathcal{F} \leftarrow \mathcal{S}$;
 4 repeat
 5     foreach $A \in$ top 3 attributes in $\mathcal{A}$ do
 6         Randomly select non-distinct value $v \in \pi_A(\mathcal{F})$;
 7         Randomly select record $r \in \mathcal{R} \setminus \mathcal{F} : \pi_A(r) = v$;
 8         $\mathcal{F} \leftarrow \mathcal{F} \cup \{r\}$;
 9     end
10 until $|\mathcal{F}| \geq s$;
11 Determine $\mathcal{H}_{\mathcal{F}}$ and $\mathcal{H}_{\mathcal{S}}$ from $\mathcal{F}$ and $\mathcal{S}$;
12 Calculate counter-balanced $\mathcal{H}_{\mathcal{SF}}$ (Equation 17);
```

Algorithm 2: Conceptual focused sampling.

should be true for any attribute. Algorithm 2 shows the focused sampling algorithm.

The algorithm starts by choosing the three attributes that are best suited for generating duplicates with equal values in the sample. Intuitively, we want attributes where some values appear more than once, but also not too often, so that an equal value may be a good indicator for a duplicate. For example, for CD records the same title or artist make a duplicate more probable.

We empirically developed the following, straight-forward scoring function, which works well on our evaluation dataset. The score maximizes for uniform distributions with an average value count of two and middle-sized string lengths:

$$score = \frac{distinct\ count}{(\varnothing count - 2)^2 + 1} \frac{1}{(\varnothing text\ length - 10)^2 + 1} \qquad (16)$$

For other datasets, users are free to define their own scoring functions, or directly select the most suitable attributes. If none of the above are feasible, we remind the reader that focused sampling is only a runtime boost, and our estimation method works also well without it.

After choosing the top three attributes, the algorithm randomly samples a fraction of the initial sample $\mathcal{S}$ (we use $\rho = .5$). It then repeatedly picks records with equal values in these attributes and adds them to the focused sample. In our implementation, we use indexes on the duplicate attribute values to efficiently pick the records in near-linear time in relation to the sample size s.

When the focused sample is complete, the algorithm calculates the duplicate size histogram, which is skewed towards larger duplicate clusters. We counter-balance the skewed histogram $\mathcal{H}_{\mathcal{F}}$ by calculating the weighted average with the histogram over the random sample $\mathcal{H}_{\mathcal{S}}$.

$$\mathcal{H}_{\mathcal{SF}} = p(v_1 = v_2)\mathcal{H}_{\mathcal{F}} + (1 - p(v_1 = v_2))\mathcal{H}_{\mathcal{S}} \qquad (17)$$

$$p(v_1 = v_2) = \frac{1}{\binom{N}{2}} \sum_{A \in top(\mathcal{A})} \sum_{v \in A} \binom{\#v}{2}$$

Here, $p(v_1 = v_2)$ is the probability to randomly choose a record pair with equal values from one of the three attributes. This probability is usually quite low ($< .01$), so that the weighted average indeed resembles the expected sampling histograms (Section 2). We now use $\mathcal{H}_{\mathcal{SF}}$ to tweak any initial histogram towards the gold histogram in a few Power iterations (10 iterations of the inner loop of Algorithm 1).

Focused sampling results in a better initial estimate $\mathcal{H}_{\mathcal{R}(0)}$ that approaches the gold histogram up to five iterations faster than the original initial estimate and therefore speeds up the overall estimation for duplicate detection runs with compute-expensive similarity measures.

5. EVALUATION

We evaluated our algorithm on three datasets with different characteristics to demonstrate the generality of our approach. We first describe the datasets and the test setup and then examine how fast our estimates converge to the gold standard with different qualities of initial histograms. The main design goal of the estimation method is to save as many candidate comparisons as possible. Hence, we are mostly interested in the number of iterations of the outer loop in Algorithm 1, since this loop enlarges the sample and causes additional comparisons. Further, we observe the variance and discuss the accuracy. Lastly, we measure the number of iterations needed to achieve certain error bounds and measure the runtime of our method.

5.1 Datasets

The **CD** dataset consists of 750,000 CD records from FreeDB with a semi-automatic gold standard [14] created by combining automatic labeling for easy-to-classify pairs and manual labeling for hard pairs. Because of its many contributors, the dataset contains 55,323 duplicates clusters, which are approximately power law distributed with cluster sizes up to 50. We use it as the main dataset for our evaluation.

A much cleaner dataset is the **Customer** dataset containing 1,039,776 person records including 89,782 artificially polluted duplicate pairs. The dataset was generated by a large industry partner to test (their) duplicate detection algorithms and simulates the integration result of three relatively clean, duplicate-free datasets with a small overlap.

On the other side of the spectrum is the **Cora** dataset with only 182 clusters in 1878 bibliographic records and a maximum cluster size of 238 records. It is a real-world dataset with a gold standard [4]. Due to the relatively huge clusters, it tests the boundaries of our method.

5.2 Test setups

For each dataset we used the corresponding gold standard to create a gold histogram. We use an oracle as the candidate comparison, which simulates a perfect similarity measure by looking up the result in the gold standard. Note, that we could have created other gold histograms with a naïve duplicate detection run with other candidate comparisons; the gold histogram would be generated with the result of that particular run. With the oracle, however, we receive the most realistic gold histogram. Further, through preliminary experiments we discovered that $\varepsilon = 10$ and $minS = 5$ show a good trade-off between accuracy and efficiency.

We repeat the estimation 100 times for each dataset with three different initial histograms and measure the estimated cluster size distribution after each iteration. In each iteration, we add $\sqrt{N}$ records from the total N records to the random sample. We choose the following three initial histograms with a maximum cluster size m. Later, we use focused sample to receive better initial histograms.

Uniform: Each record has the same probability to be in a cluster of size k. Because the cluster of size k in $\mathcal{H}^k_{\mathcal{R}(0)}$ contains k records, we normalize by $\frac{1}{k}$: $\mathcal{H}^k_{\mathcal{R}(0)} = \frac{1}{k}\frac{N}{m}$

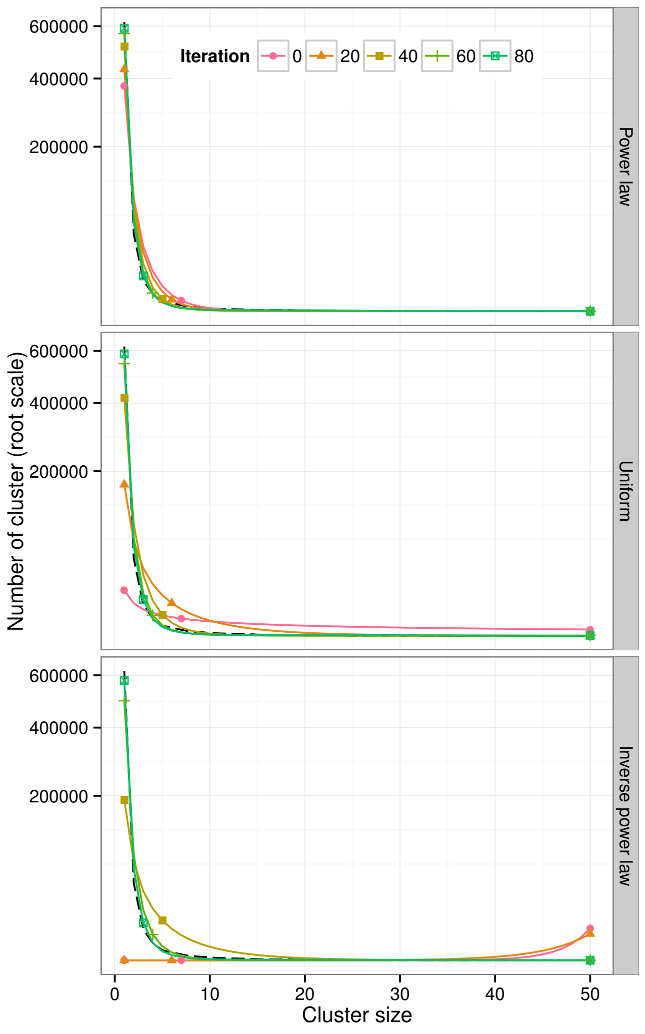

Figure 1: Convergence for 10 iterations on the CD dataset with the three different initial histograms (y-axis is square-scaled).

Power law: Half of the records are duplicate-free, a quarter is in pairs, an eights in triples and so on: $\mathcal{H}^k_{\mathcal{R}(0)} = \frac{1}{k}\frac{N}{2^k}\frac{2^m}{2^m-1}$. This is a realistic distribution for most datasets.

Inverse power law: We mirror the power law: $\mathcal{H}^k_{\mathcal{R}(0)} = \frac{1}{k}\frac{N}{2^{m-k+1}}\frac{2^m}{2^m-1}$. This is a highly unrealistic distribution, employed to test the robustness of our approach.

5.3 Different initial histogram

First, we measure the convergence of the estimation on the CD dataset in Figure 1 with the three different initial vectors. Note that we use square scales on the y-axis to better observe the relatively rare, larger cluster sizes. For further improved readability, we plotted in this and following figures only a few data points; the corresponding line is generated from data points for each cluster size.

If we use the power law distribution (top), the estimate converges quickly to the gold histogram within ten iterations. Early iterations already result in good approximations, because the initial vector corresponds well to the gold histogram.

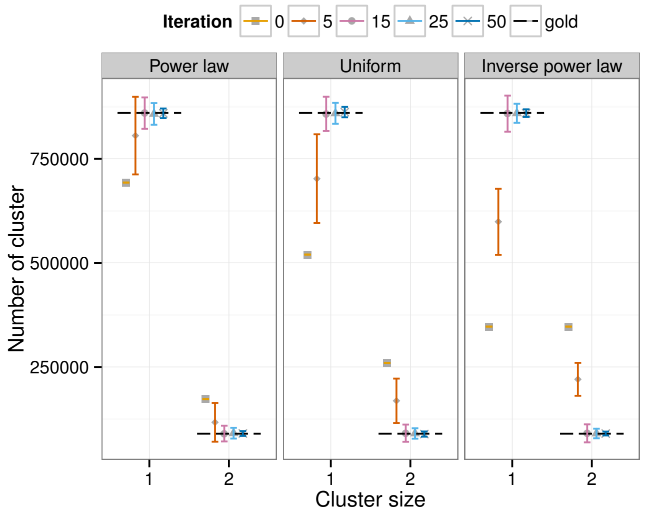

Figure 2: Standard deviation of the estimates for duplicates and non-duplicates in the Customer dataset.

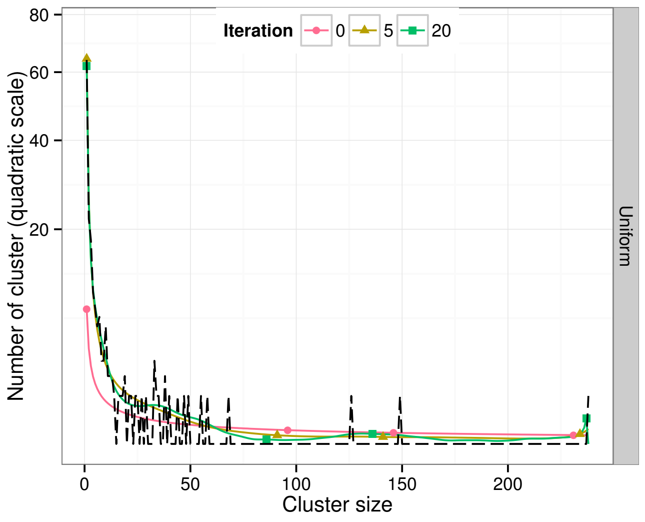

Figure 3: Convergence for 20 iterations on the Cora dataset. (y-axis square-scaled to highlight larger cluster sizes)

The uniform distribution (middle) needs 12 iterations to converge. After two iterations, we can already see a power law distribution, which then quickly approaches the gold histogram.

The estimation algorithm needs five iterations to change the inverse power law (bottom) into a power law and a total of 15 iterations to converge to the gold histogram. Nevertheless, we want to emphasize that the algorithm still converges with a sample of only about 17,000 records (2.3%) to the correct result despite the completely wrong initial estimate.

Altogether, we can see that the algorithm reliably converges. The better the initial estimate, the faster the convergence occurs. For the Cora and Customer dataset, we observe similar convergence after 15 iterations. We also conducted a sanity check and used the gold histogram as the initial histogram and observed that the estimates do not diverge from the gold histogram.

5.4 Variance of sampling

In the next experiment, we examine how the estimates vary throughout the 100 aggregated runs on the Customer dataset. This dataset contains only two cluster sizes (namely sizes 1 and 2), which allows a clear visualization of variances.

Figure 2 shows the standard deviation of the estimates at a specific iteration for the non-duplicates and the duplicate pairs and the three initial histograms. The average estimate quickly converges to the gold histogram within 15 iterations. From that point, the standard deviation decreases with additional iterations and is only ≈ 1% after 50 iterations.

5.5 Exactness of results

The first two experiments indicate that our algorithm iteratively fits the estimation to the gold histogram. We now want to closely examine the estimates of the irregular histogram of the Cora dataset. The histogram exhibits a comparably flat power law-like shape with clear peaks between cluster sizes 30 to 60 as well as peaks (individual clusters) at 127, 148, and 238.

We see that after five iterations, the curve already approaches to the overall form of the distribution in Figure 3. In successive iterations the estimate is further refined until also the individual peaks at cluster sizes 127, 148, and 238 are approximated. Please note that the y-axis is again square-scaled, so that we can examine the low frequencies of 1-5 appropriately.

Obviously, our algorithm fails to exactly determine the cluster sizes even after 20 iterations on this dataset. Nevertheless, for most use cases, the estimate after five iterations may be enough: The power-law distribution is estimated well enough and there is at least one cluster with a size over 100, which may either indicate poor data quality or an ineffective duplicate detection configuration. Further, according to our experience, larger real-world datasets are distributed more evenly, so that the hard-to-estimate peaks are less probable.

5.6 Number of iterations

Depending on the use case, the number of iterations for an estimation may vary. In this experiment, we determine a good number of iterations to meet different error bounds.

We use two measures to assess the difference between the current estimation and the gold histogram. The *root mean square error* (RMSE) calculates the normalized squared error between the estimate of each cluster size and the actual value. Because of the power law distributions, this measures heavily favors smaller, more frequent cluster sizes.

$$rmse(\mathcal{H}_\mathcal{G}, \mathcal{H}_\mathcal{R}) = \sqrt{\sum_i (\mathcal{H}^i_\mathcal{G} - \mathcal{H}^i_\mathcal{R})^2} \tag{18}$$

The *earth mover's distance* (EMD) calculates the number of records that is needed to transform one histogram to another by moving records to adjacent fields. Intuitively, the distance does not penalize slight misestimations as strongly as the RMSE. For example, if a cluster of size 237 is estimated instead of 238, the distance is at most $2 \cdot 237$, while RMSE would quadratically penalize both the present cluster at size 237 and the missing cluster at 238. We define both measures over a gold histogram $\mathcal{H}_\mathcal{G}$ and the current $\mathcal{H}_\mathcal{R}$.

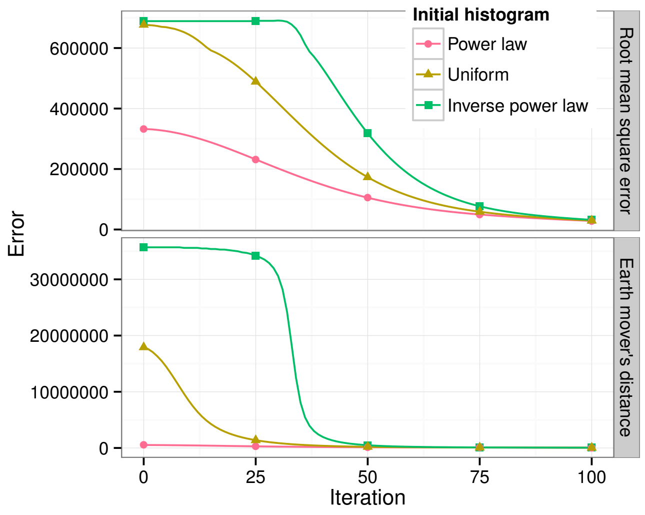

Figure 4: Root mean square error and earth mover's distance for up to 100 iterations on the CD dataset.

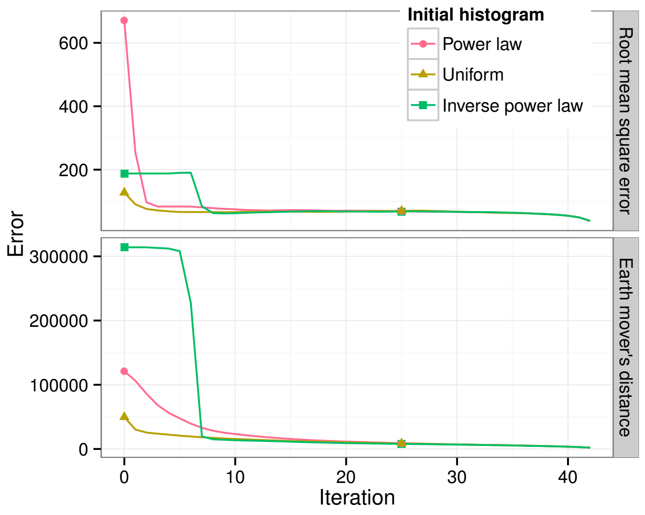

Figure 5: Root mean square error and earth mover's distance for 100 iterations on the Cora dataset.

$$emd(\mathcal{H}_\mathcal{G}, \mathcal{H}_\mathcal{R}) = \sum_i emd_i(\mathcal{H}_\mathcal{G}, \mathcal{H}_\mathcal{R}) \tag{19}$$

$$emd_i(\mathcal{H}_\mathcal{G}, \mathcal{H}_\mathcal{R}) = \begin{cases} 0 & \text{if } i = 0 \\ i \cdot (\mathcal{H}_\mathcal{G}^i - \mathcal{H}_\mathcal{R}^i) & \\ +emd_{i-1}(\mathcal{H}_\mathcal{G}, \mathcal{H}_\mathcal{R}) & \text{else} \end{cases}$$

Figure 4 shows the mean errors for up to 100 iterations on the CD dataset. The root mean square error steadily approaches zero after an initial phase that adjusts the general form of the distribution. For a good initial estimate, the error immediately decreases, while the estimate needs up to ten iterations to correct worse initial estimate.

Similarly, the earth mover's distance reflects the correction of the general form for the inverse power law distribution. For the uniform distribution the distance decreases earlier, because the shifting of the elements towards smaller clusters immediately begins. The power law distribution exhibits a low EMD from the start, since only few records need to be adjusted.

If users are not confident in their initial estimate, ten iterations are needed to guarantee a good fit of the distribution – as seen in the EMD plots. The RMSE reveals that fine-grained estimates need up to 50 iterations on the CD dataset. For better initial estimates (e.g., uniform distribution), the general form is found already after five iterations. Fine-grained estimates, however, need almost the same number of iterations for good and poor initial histograms.

The RMSE and EMD curves of the difficult Cora dataset in Figure 5 show a step decrease of error similar to the EMD curve of the CD dataset. For this dataset, the uniform distribution shows the smallest error and converges fastest. The power law surprisingly exhibits the greatest RMSE, because the number of non-duplicates is heavily overestimated (900 assumed but only 64 present). Nevertheless, the power law still represents the general form well, so that it approaches the error of the uniform distribution within three iterations. The algorithm corrects the general form of the inverse power law distribution in the seventh iteration, but keeps peaks for the large clusters and thus achieves even smaller errors than the uniform distribution.

For the Customer dataset, only two kinds of errors are possible and, thus, both RMSE and EMD are much smoother and start to decrease to zero at the first iteration. Both curves (not shown) exhibit the same characteristics, because of the close relation of EMD and RMSE in this case.

5.7 Execution time

In the last experiment, we measure the runtime of the estimation. As before, we simulate the candidate comparison with a hash lookup in the gold standard. Hence, we measure the overhead of our random walk without performing any actual candidate comparisons. Further, we measure the *reduction ratio* [1] that is defined as the ratio of saved comparisons during estimation and the total number of all pairs:

$$reduction\ ratio = 1 - \frac{\binom{n}{2}}{\binom{N}{2}} = 1 - \frac{n(n-1)}{N(N-1)} \tag{20}$$

The single-threaded experiment ran on an i5 desktop PC with 16 GB RAM and Java 1.7. The times do not include the initial data loading phase, where all data is loaded and held in-memory. For all configurations, the runtime of the random walk with fewer than two seconds is negligible even for high number of iterations. Table 2 lists the runtime in milliseconds and the reduction ratio for the three datasets.

In general and as expected, the runtime increases with the maximum cluster size and the number of records. For the CD dataset, the three slowest parts of the algorithm took 90% of the runtime, where 40% of the time is spent on the random sampling and calculation of $\mathcal{H}_\mathcal{S}$, 10% on the calculation of $T_\mathcal{S}$, and 40% on the Power iterations (inner loop of Algorithm 1). For later iterations, the calculation of $\mathcal{H}_\mathcal{S}$ and $T_\mathcal{S}$ take relatively more time, because the sample becomes larger. In contrast, the number of Power iterations linearly decreases from 100 to 25. Nevertheless, the overall time per iteration increases linearly. Hence, performing rough estimations on small samples is relatively cheap.

Dataset	Number of iterations	Runtime (in ms)	Reduction ratio
CD	10	83	99.987%
	25	190	99.917%
	50	396	99.667%
	100	1045	98.667%
Customer	10	27	99.990%
	25	53	99.940%
	50	152	99.760%
	100	632	99.040%
Cora	10	797	94.767%
	20	1104	79.043%
	30	1666	52.828%

Table 2: Runtime and relative number of comparisons for a given number of iterations per dataset.

The reduction of comparisons for the two larger datasets is enormous and solely depends on the number of iterations. As shown in the last section, coarse-grained estimation requires fewer than ten iterations and needs to perform only every 10,000th comparison. Of course, candidate selection techniques also save comparisons significantly, but as we show in Section 6, we can combine both techniques to further improve the efficiency and accuracy.

Finally, the performance can be easily improved. For the Cora dataset, the calculation of T_S dominates the overall time, but can be sped up with precalculated tables. For a product, $\mathcal{H}_S$ should be incrementally calculated. Nonetheless, the potential of our method already becomes apparent: If one assumes that one comparison takes 20 microseconds (measured in previous experiments [14]), a full naïve run needs half a year and a single-pass SNM with $w = 100$ 1 hour, while a rough estimate in ten iterations take only 45 seconds, and a fine-grained estimate with near-perfect results in 50 iterations takes less than 4 minutes.

Focused sampling. For the CD dataset, focused sampling took 10 seconds and sped up the convergence between one iteration for power law and five iterations for the inverse power law distribution. Saving one iteration reduces the estimation by 9 seconds. Therefore, focused sampling amortizes for fine-grained estimation with a higher number of iterations or more expensive candidate comparisons. Further, it quickly pays off for repeated estimation with different duplicate detection settings, since the focused sampling is performed only once and could be run in the background.

6. CANDIDATE SELECTION, MULTIPLE PASSES, AND CLUSTERING

The random walk algorithm performs naïve duplicate detection over the Cartesian product of the sample. If the actual duplicate detection run encompasses candidate selection techniques, we can see two effects. First, some duplicate pairs are not found and therefore cluster sizes become smaller and the number of clusters increases (potentially as non-duplicate clusters with single elements). Second, the runtime of the duplicate detection run decreases. Therefore, our estimate technique must also be more efficient.

Both effects can be used to improve the algorithm. The obvious solution is to calculate the sample histogram using the same candidate selection techniques, so that the estimations become more accurate for the configuration and faster at the same time. However, small adjustments to some candidate selection techniques are necessary. Further, to avoid pruning too many duplicate pairs with the candidate selection techniques, usually multiple passes are run. We discuss the most common candidate selection techniques as follows.

(Overlapping) blocking [11] groups records by some blocking key and naïvely compares the records within the groups. Duplicates can be found only within these groups. Therefore, the estimation for blocking uses the same blocking keys and analogously looks for the duplicates within the blocks. Index structures speed up the insertion of new sample records over iterations.

Sorted neighborhood method (SNM) [7] sorts the records and compares the records only within a certain window size. The estimation for SNM uses sort-merge to extend the sample using the same sorting keys. The window size w should be linearly adjusted to the sample rate with $\tilde{w} = max(2, \lceil w\frac{n}{N}\rceil)$, so that a comparable amount of neighbors are selected.

Multiple passes should be analogously applied to the sample as well. The estimation algorithm maintains multiple index structures or sorted lists. Further, duplicate comparisons can be avoided with a cache data structure.

Results from SNM or multiple passes need to be post-processed with a clustering algorithm to produce transitively closed, sane results. Our estimation implicitly assumes the transitive closure to be performed to cluster the results because of the random sample model.

Advanced **clustering algorithms**, such as CENTER [6] and MERGE-CENTER [5], split large, inhomogeneous clusters into smaller homogeneous clusters. Hierarchical algorithms may produce much smaller clusters on a sample if root records are missing in the sample. Coherent clustering algorithms may create relative large clusters on a sample, because coherence conditions depending on the size of clusters are more likely to be met in a sample. A proper adjustment of these clustering algorithm for our duplicity estimation problem is interesting future work.

7. RELATED WORK

While – to the best of our knowledge – our problem is new, similar problems have been addressed in related work.

7.1 Estimating transitive closure sizes

Our problem is related to estimating the transitive closure of graph data [3, 10]. In contrast to our problem, the (directed) edges between nodes are explicitly given.

Lipton and Naughton estimate the size of a generalized transitive closure with a basic urn model similar to our sampling model [10]. Their algorithm estimates the total number of resulting edges in near-linear time with linearly decreasing error. Similarly to our approach, the algorithm adaptively samples more records to improve the estimate until a given time limit is met. However, for our use cases the estimate of edges(=duplicate pairs) is too coarse-grained.

Cohen proposes a more stable, linear algorithm to estimate the total numbers of edges [3]. The Monte Carlo algorithm requires *reversible* edges, but can also be extended to estimate the individual neighborhoods of specific nodes, which can be used to estimate a histogram as we do.

However, both algorithms require the edges to be present, which would correspond to a full duplicate detection run in our case. While the lookup for one edge is negligible in their settings, it is an expensive candidate comparison in our case. Therefore, these techniques are not applicable as we actively try to reduce the number of candidate comparisons.

Similarity join and group-by

Duplicate detection can be viewed as a generalized similarity join if duplicate pairs are of main interest or a generalized similarity group-by if the result should be clustered.

Silva et al. define a similarity group-by operator for databases and sketch a small change to the cardinality estimation of a relational group-by operator to correctly estimate the number of groups in a similarity group-by [13]. However, the technique is not applicable to duplicate detection that uses several attributes, because a lower bound for one specific attribute often leads to low thresholds and a high overhead, while multi-attribute similarity statistics are too costly to be built and maintained.

Lee et al. use locality sensitive hashing to estimate the result size of a similarity join with cosine similarity [9]. The approach would be directly applicable to our problem if the candidate comparison consists only of one cosine similarity and threshold. However, we want to treat the candidate comparison as a black box to support a large range of duplicate detection algorithms and configurations.

Other duplicate detection techniques

Candidate selection techniques reduce the runtime and deliver near-complete results [7, 11]. Our estimation may incorporate these techniques as shown in Section 6 and thus run more accurately and efficiently.

One goal of this paper is to provide users of duplicate detection systems a feedback of their parameter settings to help improve them. Our system can be easily combined with other interactive systems to improve the user experience. Active learning incrementally tweaks a learned candidate comparison function to increase the quality [12]. Chaudhuri et al. propose an example-driven automatic assembly of a candidate selection query [2]. Both systems may be extended with our method to show users the global implications of changed settings. Further, with our focused sampling method, we can early return potentially large clusters that may indicate suboptimal settings.

8. CONCLUSION

In this paper, we defined and solved the problem of *duplicity estimation*, which estimates the cluster size distribution of a complete duplicate detection run while performing only a fraction of the candidate comparisons.

First, we developed a general sampling model that calculates the expected histogram given the histogram of a complete dataset and a sample size. Second, we devised a random walk approach that reliably and efficiently estimates the original histogram from an increasing sample. Because the original histogram is the principal eigenvector of our transition matrix, our algorithm is guaranteed to converge. Third, we proposed a focused sampling approach that finds a good initial histogram, so that the random walk algorithm converges faster.

We extensively evaluated our approach and could verify the convergence even with poor initial estimates. Better initial estimates indeed lead to faster convergence. The runtime of our approach is very low and the comparison reduction is significant. Further, we discussed how established candidate selection techniques can be incorporated to further increase the efficiency.

In future research, we plan to combine our method with more sophisticated clustering algorithms, so that the quality of the estimates in these settings is comparable to the evaluated settings.

9. REFERENCES

[1] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive Blocking: Learning to Scale Up Record Linkage. In *Proceedings of the International Conference on Data Mining series (ICDM)*, pages 87–96, 2006.

[2] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik. Example-driven design of efficient record matching queries. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 327–338, 2007.

[3] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441–453, 1997.

[4] U. Draisbach and F. Naumann. DuDe: The Duplicate Detection Toolkit. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, 2010.

[5] O. Hassanzadeh and R. J. Miller. Creating probabilistic databases from duplicated data. *VLDB Journal*, 18(5):1141–1166, 2009.

[6] T. H. Haveliwala, A. Gionis, and P. Indyk. Scalable techniques for clustering the web. In *Proceedings of the ACM SIGMOD Workshop on the Web and Databases (WebDB)*, pages 129–134, 2000.

[7] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 127–138, 1995.

[8] A. N. Langville and C. D. Meyer. Survey: Deeper Inside PageRank. *Internet Mathematics*, 1(3):335–380, 2003.

[9] H. Lee, R. T. Ng, and K. Shim. Similarity join size estimation using locality sensitive hashing. *Proceedings of the VLDB Endowment*, 4(6):338–349, 2011.

[10] R. J. Lipton and J. F. Naughton. Estimating the size of generalized transitive closures. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 165–171, 1989.

[11] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[12] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the ACM International Conference on Knowledge discovery and data mining (SIGKDD)*, pages 269–278, 2002.

[13] Y. N. Silva, W. G. Aref, and M. H. Ali. Similarity Group-By. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 904–915, 2009.

[14] T. Vogel, A. Heise, U. Draisbach, D. Lange, and F. Naumann. Reach for gold: An annealing standard to evaluate duplicate detection results. *Journal of Data and Information Quality*, 5(1–2), 2014.