Parallel Privacy-preserving Record Linkage using LSH-based Blocking

Martin Franke, Ziad Sehili and Erhard Rahm
Database Group, University of Leipzig, Germany

Keywords: Record Linkage, Privacy, Locality Sensitive Hashing, Blocking, Bloom Filter, Apache Flink.

Abstract: Privacy-preserving record linkage (PPRL) aims at integrating person-related data without revealing sensitive information. For this purpose, PPRL schemes typically use encoded attribute values and a trusted party for conducting the linkage. To achieve high scalability of PPRL to large datasets with millions of records, we propose parallel PPRL (P3RL) approaches that build on current distributed dataflow frameworks such as Apache Flink or Spark. The proposed P3RL approaches also include blocking for further performance improvements, in particular the use of LSH (locality sensitive hashing) that supports a flexible configuration and can be applied on encoded records. An extensive evaluation for different datasets and cluster sizes shows that the proposed LSH-based P3RL approaches achieve both high quality and high scalability. Furthermore, they clearly outperform approaches using phonetic blocking.

1 INTRODUCTION

Nowadays large amounts of person-related data is stored and processed, e. g., about patients or customers. For a comprehensive analysis of such data it is often necessary to link and combine data from different data sources, e. g., for data integration in health care or business applications (Christen, 2012).

The linkage and integration of data from multiple sources is challenging, especially for person-related data. Records from different databases referring to the same person have to be identified. Because of the lack of global identifiers this can only be achieved by comparing available quasi-identifiers, such as name, address or date of birth. Moreover, person-related data contains sensitive information so that a high degree of privacy should be ensured, especially if required by law (Vatsalan et al., 2017). This can be achieved by privacy-preserving record linkage (PPRL) approaches that identify records from different data sources referring to the same person without revealing personal identifiers or other sensitive information. PPRL is confronted with the Big Data challenges, particularly high data *volumes* and different data representations and qualities (*variety*, *veracity*). Hence, the key challenges for PPRL include (Vatsalan et al., 2013):

Privacy. The protection of privacy is crucial for the entire PPRL process. To fulfill this requirement it is necessary to encode the records and to conduct the linkage on the encoded data. The encoding has to preserve data properties needed for linkage and should enable efficient similarity calculations.

Quality. The identification of matching records is hindered by heterogeneous, erroneous, outdated and missing data (Christen, 2012). To achieve high match quality approximate matching techniques that can compensate data quality problems are necessary.

Scalability. PPRL should scale to millions of records from two or more data owners (parties). The trivial approach is to compare every possible record pair from the different data sources. For two parties this lead to a quadratic complexity depending on the size of the databases to be linked. To make PPRL scalable to large datasets blocking and filtering techniques are used. Another option is to perform PPRL in parallel on multiple processors.

The aim of this work is to improve the scalability and overall performance of PPRL by supporting both parallel PPRL (P3RL) and blocking. Our P3RL approach enables the utilization of large shared nothing clusters running state-of-the-art distributed processing frameworks, such as Apache Flink[1] or Apache Spark[2], to reduce the execution time proportional to the number of processors in the cluster. Our P3RL approaches utilize blocking to partition the records such that only records within the same block need to be compared. These comparisons are performed in

[1]https://flink.apache.org/
[2]https://spark.apache.org/

Franke, M., Sehili, Z. and Rahm, E.
Parallel Privacy-preserving Record Linkage using LSH-based Blocking.
DOI: 10.5220/0006682701950203
In *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018)*, pages 195-203
ISBN: 978-989-758-296-7

parallel by distributing the blocks among all worker nodes within the cluster. In this paper, we focus on blocking based on locality sensitive hashing (LSH) (Indyk and Motwani, 1998; Durham, 2012) which can be applied on encoded data and is not domain-specific. To comparatively evaluate the efficiency of LSH in a parallel setting we further developed a modified phonetic blocking approach based on Soundex (Odell and Russell, 1918). We parallelize both approaches and compare them in terms of quality, efficiency and scalability. Following previous work, we realize our P3RL approach as three-party protocol using a trusted third party to conduct the linkage, the linkage unit (LU) (Vatsalan et al., 2013). The use of such a LU is well-suited for P3RL because the LU can maintain a high-performance cluster. As privacy technique we adopt the widely-used method by Schnell (Schnell et al., 2011) to encode record values with Bloom filters (Bloom, 1970).

Specifically, we make the following contributions:

- We develop parallel PPRL (P3RL) approaches with LSH-based and phonetic blocking using a state-of-the-art distributed processing framework to efficiently execute PPRL on large-scale clusters. For LSH blocking we include optimizations such as to avoid redundant match comparisons.
- We comprehensively evaluate the quality, efficiency, scalability and speedup of our P3RL approaches for different parameter settings and large datasets with up to 16 million records in a cluster environment with up to 16 worker nodes.

After a discussion of related work in the next section, we describe preliminaries regarding PPRL and LSH-based blocking. In Sec. 4, we present our P3RL approaches using Apache Flink for both LSH and phonetic blocking. In Sec. 5, we evaluate the new approaches for different datasets and cluster sizes. Finally, we conclude our work.

2 RELATED WORK

Record Linkage (RL). The problem of finding records that represent the same real-word entity has been studied for over 70 years (Christen, 2012). RL approaches aim at achieving a high match quality and scalability to large datasets (Köpcke and Rahm, 2010). To reduce the number of record comparisons blocking techniques are used. The standard blocking method defines a blocking key (BK) to group records into blocks such that only records of the same block are compared (Christen, 2012). A BK is determined by applying a function (e. g., Soundex or attribute substring) on one or more selected record attributes (Fisher et al., 2015). For example, one could block persons based on the Soundex value of their last name (phonetic blocking) or on the concatenation of the initial two letters of their first name and the city of birth.
Privacy-preserving Record Linkage (PPRL). The additional requirement to preserve the privacy of entities led to the development of techniques that allow a linkage based on encoded values. Analogous to RL, blocking techniques are applied to make PPRL scalable to large datasets. A common approach is to use phonetic codes to enable blocking based on phonetic similarities of attribute values (Karakasidis and Verykios, 2009). LSH-based blocking has shown to achieve high match quality as well as scalability (Durham, 2012; Karapiperis and Verykios, 2015; Karapiperis and Verykios, 2016).
Parallel Record Linkage (PRL). For a further improvement of the scalability several parallelization techniques have been considered for RL. On the one hand graphic processor have been used to speed up the similarity computations (Forchhammer et al., 2013; Ngomo et al., 2013). Another approach is to utilize Hadoop MapReduce as parallel processing framework to conduct the linkage in parallel (Wang et al., 2010; Dal Bianco et al., 2011; Kolb et al., 2012).
Parallel Privacy-preserving Record Linkage (P3RL). We are aware of only two studies on parallel approaches for PPRL. In (Sehili et al., 2015), graphic processors are utilized for a parallel matching of Bloom filters (bit vectors). Furthermore, (Karapiperis and Verykios, 2013) and (Karapiperis and Verykios, 2014) proposed the use of MapReduce to improve the scalability of Bloom-filter-based PPRL with LSH-based blocking. As in the MapReduce approaches for RL (Kolb et al., 2012), the map function is used to determine the BK values (LSH keys) in parallel. Then, the records are grouped based on their LSH keys and block-wise compared in the reduce step. To address the problem of duplicate candidate pairs in different blocks, two MapReduce jobs are chained. While the first job only emits the ids of candidate record pairs, the second job groups equal candidate pairs in the reduce step to calculate the similarity of each pair only once. Here the usage of MapReduce shows some limitations: In contrast to modern distributed processing frameworks like Apache Flink, MapReduce does not support complex user-defined functions and requires expensive job chaining and other workarounds instead. Moreover, the evaluation is limited to two and four nodes and small datasets of only about $300,000$ records. As a result, the scalability of the approach to larger datasets with millions of records and larger clusters remains open.

3 PRELIMINARIES

3.1 Bloom Filter Encoding

A Bloom filter (BF) (Bloom, 1970) is a bit vector of fixed size **m** where initial all bits are set to 0. BFs are used to represent a set of elements $E = \{e_1, \dots, e_\omega\}$. **k** independent cryptographic hash functions $h_1, \dots, h_k$ are selected. For an element $e \in E$ each hash function h_i with $1 \leq i \leq k$ produces a position $p_i \in [0, m-1]$. The bits at the resulting k positions are then set to 1.

BFs are widely used for PPRL to encode records with their attribute values (Vatsalan et al., 2013). Several approaches for constructing BFs have been proposed to reduce the re-identification risk and improve the linkage quality (Schnell et al., 2011; Durham, 2012; Vatsalan et al., 2014). We use the approach proposed in (Schnell et al., 2011) where the values of selected record attributes are tokenized into a set E' of Q-grams (substrings of length **Q**) and then hashed into one single BF, the cryptographic long-term key (CLK). To determine the similarity of two BFs binary similarity measures, mainly the Jaccard or Dice similarity, are used (Vatsalan et al., 2013). If two BFs have a similarity value equal or above a predefined threshold $\mathbf{t} \in [0, 1]$ the BF pair is classified as match.

An appropriate choice of m and k is essential, since BFs can return false-positives. The optimal BF size m_{opt} depends on k and the number of considered Q-grams $\omega = |E'|$ such that $m_{opt} = \lceil (k \cdot \omega)/\ln(2) \rceil$ (Mitzenmacher and Upfal, 2005). Because m is fixed for all BFs, ω is calculated as avg. number of Q-grams over all n records r_i with $1 \leq i \leq n$ to be linked, such that $\omega_{avg} = \lceil (\sum_{i=1}^{n} |E'_i|)/(n) \rceil$, where $|E'_i|$ denotes the number of Q-grams a record r_i produces.

BFs are susceptible to cryptanalysis because frequent Q-grams lead to frequent 1-bits. Several studies have analyzed attacks on BFs (Kuzu et al., 2011; Kuzu et al., 2013; Kroll and Steinmetzer, 2014; Niedermeyer et al., 2014; Christen et al., 2017). However, these attacks are only feasible for certain assumptions and building methods, especially for field-level BFs (Schnell et al., 2009). For record-level BFs, such as CLK, the re-identification risk can be reduced by applying hardening techniques (Niedermeyer et al., 2014; Schnell, 2015; Schnell and Borgs, 2016).

3.2 Locality Sensitive Hashing

LSH was proposed to solve the nearest neighbor problem in high-dimensional data spaces (Indyk and Motwani, 1998). For LSH a family of hash functions $\mathcal{F}$ that is sensitive to a distance measure $d(\cdot,\cdot)$ is used. Let d_1, d_2 with $d_1 < d_2$ be two distances according to d; pr_1, pr_2 with $pr_1 > pr_2$ two probabilities and Z a set of elements. A family $\mathcal{F}$ is called (d_1, d_2, pr_1, pr_2)-*sensitive* if for all $f \in \mathcal{F}$ and for all elements $x, y \in Z$ the following conditions are met:

- $d(x,y) \leq d_1 \quad \Rightarrow \quad \mathbb{P}(f(x) = f(y)) \geq pr_1$
- $d(x,y) \geq d_2 \quad \Rightarrow \quad \mathbb{P}(f(x) = f(y)) \leq pr_2$

With that the probability that a function $f \in \mathcal{F}$ returns the same output for two elements with a distance smaller or equal to d_1 is at least pr_1. Otherwise, if the distance is greater or equal to d_2 then the probability that f returns the same output is at most pr_2.

For applying the LSH method in PPRL context, the two hash families approximating the Jaccard and the Hamming distance are most relevant (Durham, 2012). We focus on the hash family $\boldsymbol{\mathcal{F}_{\mathcal{H}}}$ that is sensitive to the Hamming distance (HLSH). Each function $f_i \in \mathcal{F}_{\mathcal{H}}$ with $0 \leq i \leq m-1$ maps a BF Bf_j representing a record r_j to the bit value on position i of Bf_j. For LSH-based blocking, a set $\Theta = \{f_{\lambda_1}, \dots, f_{\lambda_\Psi} \mid f_{\lambda_\iota} \in \mathcal{F}\}$ of $\boldsymbol{\Psi}$ hash functions is used. To group similar records, a blocking key $BK_\Theta(Bf_j)$ is generated by concatenating the output values of the hash functions $f_\lambda \in \Theta$, such that $BK_\Theta(Bf_j) = f_{\lambda_1}(Bf_j) \odot \dots \odot f_{\lambda_\Psi}(Bf_j)$, where $\odot$ denotes the concatenation of hash values. As a BK consists of $\Psi < m$ function values, Ψ defines the length of the blocking (LSH) key. Based on the probabilistic assumption of LSH, the BKs of two similar BFs with a distance smaller or equal to d_1 may be different. For this reason, $\boldsymbol{\Lambda}$ BKs $BK_{\Theta_1}, \dots, BK_{\Theta_\Lambda}$ are used to increase the probability that two similar BFs have at least one common BK.

Example: Let $\Theta_1 = \{f_7, f_1\}$, $\Theta_2 = \{f_0, f_5\}$ and $Bf_1 = \mathbf{1}1\mathbf{0}11\mathbf{0}1\mathbf{1}$, $Bf_2 = \mathbf{1}00\mathbf{1}1\mathbf{0}1\mathbf{1}$ two BFs. We get $BK_{\Theta_1}(Bf_\mathbf{1}) = f_7(Bf_1) \odot f_1(Bf_1) = 11$, $BK_{\Theta_2}(Bf_\mathbf{1}) = f_0(Bf_1) \odot f_{15}(Bf_1) = 10$ and $BK_{\Theta_1}(Bf_\mathbf{2}) = 10$, $BK_{\Theta_2}(Bf_\mathbf{2}) = 10$. Hence, Bf_1 and Bf_2 agree on BK_{Θ_2} and will put into the same block and be compared.

The parameters $\boldsymbol{\Psi}$ and $\boldsymbol{\Lambda}$ influences the efficiency and effectiveness of a LSH-based blocking approach. The higher Ψ (LSH key length) the higher is the probability that only records with a high similarity are assigned to the same block. Thus, the number of records per block will be smaller. But, a higher Ψ also raises the probability that matching records are missed due to erroneous data. Λ determines the number of BKs. Hence, a higher value of Λ increases the probability that two similar BFs have at least one common BK. However, an increasing Λ leads to more computations and can deteriorate the scalability. Basically, Λ should be as low as possible while Ψ is high enough to build as many blocks so that the search space is greatly reduced. An optimal value for Λ can analytically be determined dependent on Ψ and on the similarity of true matching BF pairs (Karapiperis and Verykios, 2014). This is difficult to utilize in practice since the similarity of true matches is generally unknown. Using HLSH Ψ should be sufficiently large because each function $f \in \mathcal{F}_{\mathcal{H}}$ can only return 0 or 1 as output. Thus at most 2^{Ψ} BK values (blocks) exist for one HLSH key.

4 PARALLEL PPRL (P3RL)

We now explain our P3RL framework based on Apache Flink. At first, we give a brief introduction to Flink and outline basic concepts of our framework. We then describe the implementation of the PPRL process using HLSH and phonetic blocking. For our HLSH-based blocking approach we propose two optimizations which aim at avoiding duplicate match comparisons and restrict the choice of HLSH keys by avoiding the most frequent 0/1-bit positions.

4.1 Apache Flink

We based our implementation on Apache Flink (Carbone et al., 2015) which is an open-source framework for the in-memory processing of distributed dataflows. Flink supports the development of programs that can be automatically executed in parallel on large-scale clusters. A Flink program is defined through *Streams* and *Transformations*. Streams are collections of arbitrary data objects. Since we have a fixed set of input records, we use bounded streams of data, called *DataSets*, and the associated *DataSet API*. Transformations produce new streams by modifying existing ones. Multiple transformations can be combined to perform complex user-defined functions. Flink offers a wide range of transformations, where some of them are adopted from the MapReduce paradigm (Dean and Ghemawat, 2008).

4.2 General Approach

The general approach of our distributed framework using Flink is illustrated in Figure 1. Similar as in former work, at first each party individually conducts a preprocessing step where records are encoded and static BKs can be defined (Vatsalan et al., 2013). Then, the parties send their encoded records as BFs to the LU. The LU utilizes a HDFS cluster and stores the BFs distributed and replicated among the cluster nodes. To conduct the linkage, the data is read in parallel by the nodes. If the BFs do not contain a BK the LU generates them. Afterwards, the blocking step is conducted to group together similar records for search space reduction. Hence, the records are distributed and redirected among the cluster nodes based on their BKs. Thereby, all records with the same BK are sent to the same worker. Finally, the workers build candidate pairs, optionally remove duplicates and perform the similarity calculations in parallel. The IDs of the matching pairs are then sent to the data owners.

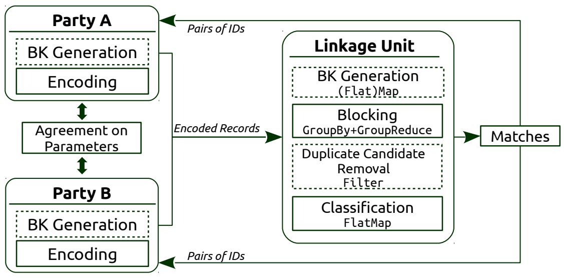


Figure 1: General approach of the P3RL using Flink. A dotted line indicates an optional step.

4.3 Hamming LSH

The first step of our HLSH blocking approach is to calculate the BKs $BK_{\Theta_1}, \ldots, BK_{\Theta_\Lambda}$ for every input BF Bf_i within a `FlatMap` function. Such function applies a user-defined `Map` function to each record and returns an arbitrary number of result elements. We choose $f_{\lambda_1}, \ldots, f_{\lambda_\Psi}$ randomly from $\mathcal{F}_{\mathcal{H}}$ for each BK, but using each $f_{\lambda_\iota} \in \mathcal{F}_{\mathcal{H}}$ only once so that each HLSH key uses different bit positions. More formally, we set $\Theta_x \cap \Theta_y = \emptyset \ \ \forall x, y \in \{1, \ldots, \Lambda\}$. The output of the `FlatMap` function are tuples of the form (keyId, keyValue, record). Each Bf_i is replicated Λ times since it produces a tuple $T_j^i = (j, BK_{\Theta_j}(Bf_i), Bf_i)$ for every $j \in \{1, \ldots, \Lambda\}$. The first two fields of T correspond to the HLSH key with its id and the third field consists of the input BF.

Then, on the first two fields of each T_j^i a `GroupBy` function is applied. By that the tuples are redistributed so that tuples with the same HLSH key value are assigned to the same block and worker. By using a `GroupReduce` function, every pair of tuples within a block is formulated as candidate pair $C_j^{i',i''} = (Bf_{i'}, Bf_{i''})$ if the BFs originate from different parties. A `GroupReduce` function is similar to a `Reduce` function but it gets the whole group (block) at once and returns an arbitrary number of result elements.

Finally, the similarity of all candidate pairs is computed within a `FlatMap` function to output only candidates with a sufficient similarity value. Afterwards, the matches are written into the HDFS.

4.3.1 Removal of Duplicate Candidate Pairs

By using multiple BKs LSH generates overlapping blocks. Consequently, pairs of BFs may occur in multiple blocks so that they are compared several times. To avoid these redundant similarity calculations, we adapted the approach from (Kolb et al., 2013), so that for every tuple T_j^i additionally a list of all HLSH keys until the $(j-1)$-th key is emitted. We get tuples $\hat{T}_j^i = (j, BK_{\Theta_j}(Bf_i), Bf_i, keys_j(Bf_i))$ with $keys_j(Bf_i) = \{BK_{\Theta_1}(Bf_i), \ldots, BK_{\Theta_{j-1}}(Bf_i)\}$. For each $\hat{C}_j^{i',i''} = ((Bf_{i'}, keys_j(Bf_{i'})), (Bf_{i''}, keys_j(Bf_{i''})))$ it is checked, if the HLSH key lists are disjoint, i. e. if $keys_j(Bf_{i'}) \cap keys_j(Bf_{i''}) = \emptyset$. If they are disjoint, then BK_{Θ_j} is the least common HLSH key and the candidate pair is compared. Otherwise, a $BK_{\Theta_{j'}}$ with $j' < j$ exists so that the candidate pair is already considered in another block and can be pruned for BK_{Θ_j}. We realized this overlap check with a `Filter` function which is applied to each candidate pair. If the function evaluates to true, i. e. the key lists do not overlap, the similarity of the candidate pair will be calculated. Otherwise, the filter function evaluates to false and the candidate pair is pruned.

The avoidance of redundant match comparisons leads to additional computational effort. At maximum $O(\Lambda - 1)$ HLSH keys need to be compared for each candidate pair. Moreover, the tuple objects are larger leading to higher network traffic.

4.3.2 HLSH Key Restriction (HLSH-KR)

HLSH uses randomly selected bits from BFs to construct BKs. By that, HLSH applies probabilistic blocking on the presence/absence of certain Q-grams in the attribute values of a record. With growing data volume some Q-grams can occur in many records, because of limited real-world designations and name spaces that can be built by linguistic units (e. g. morphemes, phonemes) of natural languages. For example, most residential addresses end with suffixes like 'street' or 'road'. Even if such suffixes are abbreviated many records will produce Q-grams like 'st' or 'rd' resulting in the same 1-bits in the BFs. Another example is the attribute gender, assuming only two possible values ('female', 'male'). If mapped into BFs, every BF will contain the same 1-bits corresponding to Q-grams resulting from the substring 'male'. If such frequently occurring 1-bits are used to construct a HLSH key, many records will share the same HLSH key and are assigned to the same block. This will lead to large blocks with records only agreeing on Q-grams having a low discriminatory power. On the other hand, bit positions where the majority of bits are 0-bits can exist. For example, some Q-grams are very rare, because they are not common or are only existing due to erroneous data (typographical errors). Again, using these bit positions for constructing HLSH keys can lead to large blocks, because many records share the property that they do not contain these information. To overcome these issues, very frequent/infrequent Q-grams could be treated as stop words to avoid them to be encoded into BFs. However, the identification of such Q-grams depends on the domain and on the used record attributes. Removing Q-grams also influences the linkage quality by changing the similarity values of BF pairs.

Based on these observations, we propose to consider only those bit positions for the HLSH keys where no frequent 0/1-bits occur. For this purpose we count the number of 1-bits for each position and for all input BFs. The resulting list of bit positions is sorted w. r. t. the number of 1-bits at the corresponding position in ascending order. Then, we remove $\frac{1}{v}$ bit positions at the beginning (frequent 0-bits) and at the end of the list (frequent 1-bits) resulting in a bit position list P. Here v denotes the pruning proportion for frequent bits. For the HLSH key generation, we choose the hash functions randomly from $\tilde{\mathcal{F}}_{\mathcal{H}} = \{f_\iota \in \mathcal{F}_{\mathcal{H}} \mid \iota \in P\}$.

4.4 Phonetic Blocking

To comparatively evaluate our HLSH approach we consider phonetic blocking (PB) as a baseline for comparison. The idea of PB is to use a phonetic encoding function that produce the same output for input values with a similar pronunciation. Usually attributes like surname or given name are used to group persons with a similar name while ignoring typographical variations. For PB the BK is constructed during the preprocessing step. Each party individually builds for every record a phonetic code for a selected attribute. Providing phonetic codes as plain text reveal some information about the encoded records. For example, Soundex reveal the first letter of an attribute value thereby providing an entrance point for cryptanalysis. Therefore, we encode the phonetic code for a record r_i into a separate BF that is used as regular BK.

Since frequent/rare phonetic codes are potentially identifiable by analyzing the frequency distribution of the blocking BF values, we consider a second variant denoted as *salted phonetic blocking (SPB)*. Similar to (Schnell, 2015), we select a record specific key used as salt for the BF hash functions to affect the hash values and corresponding BF bit positions. If the salting keys of two records differs, it is very unlikely that the same phonetic code will produce the same BF. Consequently, the attribute(s) used as salting key should be flawless because otherwise many false-negatives will occur. Following this idea, for SPB we use a second phonetic code as salt such that each hash function gets as input both phonetic codes concatenated. Since only records agreeing in both phonetic codes are assigned to the same block, the number of blocks increases and the block sizes decreases.

5 EVALUATION

In this section, we evaluate our P3RL approaches in terms of quality, scalability and speedup. Before presenting the evaluation results we describe our experimental setup and the datasets and metrics we used.

5.1 Experimental Setup

We conducted our experiments using a cluster with 16 worker nodes. Each worker is equipped with an Intel Xeon E5-2430 CPU with 6×2.5 GHz, 48 GB RAM, two 4 TB SATA disks running openSUSE 13.2. The nodes are connected via 1 Gbit Ethernet. We use Hadoop 2.6.0 and Flink 1.3.1. Flink is executed standalone with 1 JobManager and 16 TaskManagers, each with 6 TaskSlots and 40 GB JVM heap size.

5.2 Datasets

We generated synthetical datasets using the data generator and corruption tool GeCo (Christen and Vatsalan, 2013). We replaced the lookup files for attribute values by German names and address lists[3] and added realistic frequency values drawn from German census data[4]. To consider different PPRL scenarios we generated datasets D_S and D_R. With $\mathbf{D_S}$ a *statewide* linkage is simulated by using the complete look-up files. In contrast, we restrict the addresses for records from $\mathbf{D_R}$ to a certain region by only considering cities whose zip code starts with '04'. With that, D_R simulates a *regional* linkage scenario which imitates a linkage of patient records from local health care providers. For both datasets we build subsets D_{S_n} and D_{R_n} of size $n \cdot 10^6$ for all $n \in \{2^b \mid b \in \mathbb{N} : b \leq 4\}$. Each obtained dataset consists of $(4n/5) \cdot 10^6$ original and $(n/5) \cdot 10^6$ randomly selected duplicate records. To simulate dirty data each duplicate is corrupted by choosing α attributes where in each at maximum β modifications are inserted. We get datasets $D_{S_n}^{(\alpha,\beta)}$ and $D_{R_n}^{(\alpha,\beta)}$ respectively. We consider two levels of corruptions (*moderate* and *high*) by generating $D_{S_n}^M$, $D_{R_n}^M$ with $M = (2,1)$ and $D_{S_n}^H$ with $H = (3,2)$.

The records are encoded into BFs (CLK) by tokenizing the values of the attributes $A = \{$surname, first name, date of birth (dob), city, zip code$\}$ into a set of trigrams. We set $k = 20$. To determine m_{opt} we estimate the avg. number of trigrams $\varepsilon(a_i)$ each attribute $a_i \in A$ with an avg. length of Γ produces: $\varepsilon(a_i) = (\Gamma - Q) + 1$. Our estimation using character padding (Schnell, 2015) to build trigrams is shown in Tab. 1. Finally, we approximate ω_{avg} with $\omega_{avg} \approx \sum_{i=1}^{|A|} \varepsilon(a_i) = 42$ leading to $m_{opt} = 1212$ (see Sec. 2). For PB and SPB we choose the attributes surname and first name (salt) leading to an optimal blocking BF length of 29 using 20 hash functions.

Table 1: Estimation of the avg. attribute length and the resulting number of Q-grams.

	Name	Surname	Dob	Zip + City
Avg. field length	6	7	8	5 + 10
Q-grams (Q = 3)	8	9	10	15

5.3 Evaluation Metrics

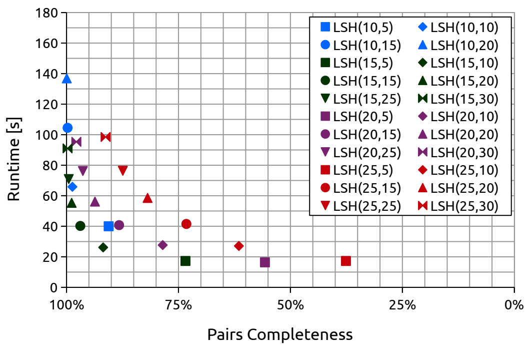
To assess the match quality of our blocking schemes, we measure the *pairs completeness (PC)* that is the ratio of true matches ($PC \in [0,1]$) that can be identified within the determined blocks (Christen, 2012). To evaluate scalability, we measure the *execution times* for several datasets of different sizes. We also calculate the *reduction ratio (RR)* which is defined as the fraction of record pairs that are removed by a blocking method compared to the evaluation of the Cartesian Product (Christen, 2012). The achievable *speedup* is analyzed by utilizing different cluster sizes (Υ).

5.4 Experimental Results

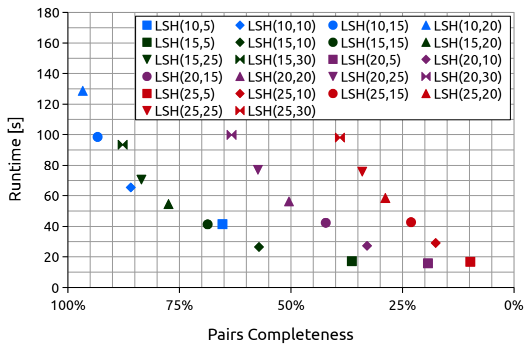
HLSH Parameter Evaluation: To analyze effectiveness and efficiency, we evaluate different HLSH parameter settings by considering several HLSH key lengths with $\Psi \in \{10,15,20,25\}$ and varying the number of HLSH keys $\Lambda \in \{5,10,15,\ldots,30\}$. For this experiment we use $D_{S_1}^M$ and $D_{S_1}^H$ (1 million records) setting $\Upsilon = 4$. Fig. 2 shows the PC values compared to the runtimes for the different HLSH settings denoted as LSH(Ψ,Λ) for both moderate and high degrees of corruption. For a moderate degree of corruption (Fig. 2a), several configurations achieve a good PC of over 95%. A high PC result is favored by low key lengths Λ (e. g., 10 or 15) and thus relatively

[3]The lookup files shipped with GeCo are very small which makes them not suitable for generating large datasets.

[4]https://www.destatis.de/DE/Methoden/Zensus_/Zensus.html



(a) $D_{S_1}^M$



(b) $D_{S_1}^H$

Figure 2: Pairs Completeness against run time of different HLSH parameter settings for $D_{S_1}^M$ and $D_{S_1}^H$ and $\Upsilon = 4$.

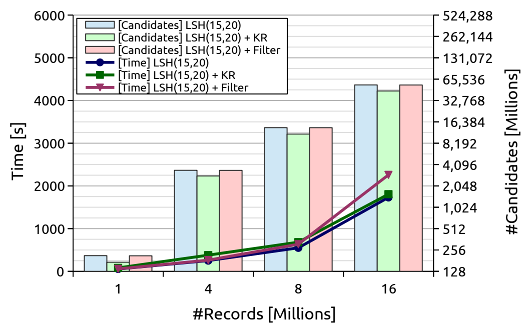
large blocks. The best results for $\Psi = 10$ require relatively high run times due to the larger block sizes compared to configurations with larger key length (since per key at most 2^{10} blocks are generated for $\Psi = 10$). A near-optimal PC with better run times is achieved with $\Psi = 15$ and $\Lambda = 20$. A large key length such as $\Psi = 25$ misses many matches and achieves only relatively low PC values even with many BKs, e. g. $\Lambda = 30$. This is even more apparent for the results with a high degree of corruption (Fig. 2b). Here, choosing $\Psi \geq 20$ achieves unacceptably low PC values of less than 65 %. The best results are achieved for $\Psi = 10$ and $\Psi = 15$ where the latter setting needs already a high number of BKs of at least 30. The best trade-off between PC and runtime can be achieved by using $\Psi = 15$. But, facing very dirty data as simulated with $D_{S_1}^H$, Λ should be sufficiently large with $\Lambda \geq 30$.

Table 2: Comparison of the blocking schemes considering PC, RR, the number of blocks (per HLSH key) and the number of candidates (in millions) for $D_{S_1}^M$ and $D_{S_1}^H$.

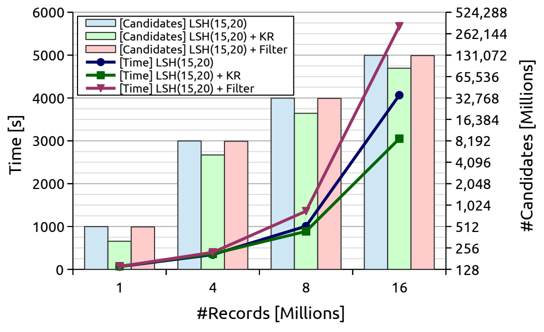
D	Method	PC [%]	RR [%]	# Blocks	# Cand.
$D_{S_1}^M$	LSH(10,10)	98.68	98.47	1,024	2,446.25
	LSH(10,15)	**99.76**	97.69	1,024	3,702.66
	LSH(15,15)	96.92	99.90	32,425	161.56
	LSH(15,20)	98.85	99.87	32,448	213.11
	PB	86.70	99.62	2,347	610.77
	SPB	73.11	**99.99**	**79,864**	**19.19**
$D_{S_1}^H$	LSH(10,10)	85.90	98.52	1,024	2,375.22
	LSH(10,15)	**93.34**	97.76	1,024	3,580.99
	LSH(15,15)	68.68	99.91	32,489	151.01
	LSH(15,20)	77.45	99.88	32,504	199.74
	PB	75.50	99.66	2,737	551.34
	SPB	54.90	**99.99**	**84,071**	**18.36**

In the following we compare the best performing HLSH settings to the phonetic blocking approaches PB and SPB. The results using the same datasets as before are shown in Tab. 2. One can see that PB and SPB achieve a quite low PC even for $D_{S_1}^M$. The reason is that phonetic blocking is is more sensitive w. r. t. data errors compared to HLSH blocking. All approaches except those with $\Psi = 10$ achieve a high RR of over 99 % and are thus able to greatly reduce the search space. However, the number of candidates varies significantly between the methods, even if the RR values are very close. For instance, the number of candidates for PB is more than a factor of 3.5 higher compared to LSH(15,15). SPB instead generates 30 times fewer candidate pairs compared to PB. This is due to the low number of blocks that are generated by PB. While Soundex theoretically can produce $26 \cdot 7^3$ BK values (blocks), less than $8 \cdot 7^3$ blocks are actually build making PB not feasible for large datasets.

HLSH Optimizations: We evaluate the HLSH optimizations described in Sec. 4.3.1 and Sec. 4.3.2 for the datasets $D_S{}^M$, $D_R{}^M$ setting $v = \frac{1}{8}$ and $\Upsilon = 4$. Fig. 3 shows the runtimes and the number of candidates (logarithmic scale on the right-side y-axis) achieved by LSH(15,20) while enabling our optimizations. In general, the number of candidates for $D_R{}^M$ is about 2.4 times as high as for $D_S{}^M$ due to a higher degree of similarity between the records and thus larger blocks. For both datasets, the removal of duplicate candidates could reduce the number of candidates very little by at most 1 %. Hence, the overhead checking the lists of HLSH keys was more significant so that the total runtimes increased significantly. By contrast, the key restriction approach HLSH-KR reduces the number of candidates substantially by up to 20 % for $D_S{}^M$ and even 40 % for $D_R{}^M$. While for $D_S{}^M$ the overhead for calculating the bit frequencies neutralize the savings, for $D_{R_8}^M$ and $D_{R_{16}}^M$ HLSH-KR leads to significant runtime savings. HLSH-KR generally improves with growing data volumes since the data space becomes more dense such that more frequent 0/1-bits occur and thus avoiding overly large blocks becomes more beneficial. The overhead to determine frequent 0/1-bits is linear and becomes negligible for large datasets because the complexity of PPRL remains quadratic. Using HLSH-KR also influences the linkage quality since fewer bit positions



(a) $D_S{}^M$



(b) $D_R{}^M$

Figure 3: Evaluation of HLSH duplicate candidate filter and HLSH key restriction (denoted as KR) for $D_S{}^M$ and $D_R{}^M$ setting $v = \frac{1}{8}$ and $\Upsilon = 4$.

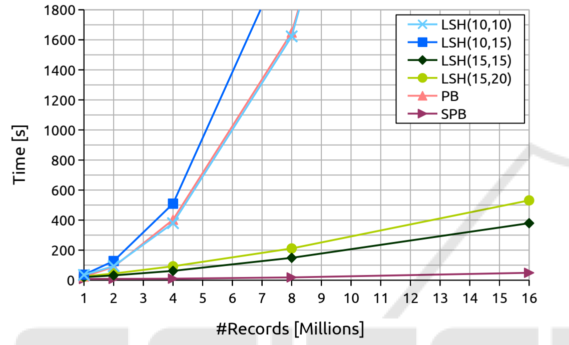


Figure 4: Execution times for different blocking methods for $D_S{}^M$ with $\Upsilon = 16$.

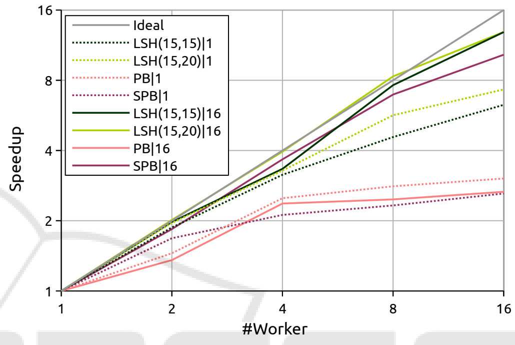


Figure 5: Speedup.

are considered. For both datasets we measured a loss of PC of around 1 % while enabling HLSH-KR.

Scalability and Speedup: To evaluate the scalability we execute our blocking approaches on the datasets $D_{S_1}^M, \ldots, D_{S_{16}}^M$ setting $\Upsilon = 16$. The runtime results depicted in Fig. 4 show that LSH(10,10), LSH(10,15) and PB do not scale w. r. t. the number of records which is due to the low number of blocks. Already for the dataset $D_{S_4}^M$, these approaches run more than 8 times slower than the other. For $D_{S_8}^M$ and $D_{S_{16}}^M$ the runtime increases drastically, so that PB for example requires around 1650 s and 7120 s, respectively. By contrast, SPB can achieve the lowest execution times, albeit for an unacceptably low PC (Tab. 2). In general, even a slightly higher RR can lead to a huge performance improvement. The LSH blocking approaches with $\Psi = 15$ are able to efficiently link large datasets with a high match quality. For example, LSH(15,20) is able to conduct the linkage of $D_{S_{16}}^M$ in around nine minutes. It is also interesting that the runtime of LSH(15,20) is higher by a factor of 1.4 compared to LSH(15,15). Hence, the increase of the runtime corresponds to the higher value for Λ ($\frac{4}{3}$). Finally, we evaluate the speedup by utilizing up to 16 worker nodes using $D_{S_1}^M$ and $D_{S_{16}}^M$. The results in Fig. 5 show that for $D_{S_{16}}^M$ and up to eight worker nodes, the speedup is nearly linear for both LSH methods, while degrades after this. In contrast, PB achieves a much lower speedup that degrades already for more than four workers. This is because of the low number of blocks (Tab. 2) and because of data skew effects introduced by frequent names leading to large blocks that need to be processed on a single worker. The speedup results for $D_{S_1}^M$ are lower compared to $D_{S_{16}}^M$ indicating that the lower data volume can be handled already by fewer workers. This is confirmed by the speedup of the SPB approach which already degrades for more than two worker nodes.

6 CONCLUSION

We proposed and evaluated parallel PPRL approaches using LSH-based blocking. We make use of Apache Flink as state-of-the-art distributed processing framework. Our evaluation for large datasets and different degrees of data quality showed the high efficiency and effectiveness of our LSH-based P3RL approaches which clearly outperform approaches based on phonetic blocking. We also found that the overhead for finding duplicate candidates cannot be outweighed by the achievable savings. By contrast, avoiding the most frequent 0/1-bits for LSH blocking proved to be beneficial for very large datasets. For future work, we plan to investigate further P3RL approaches and make them available in a toolbox for use in applications and for a comparative evaluation.

REFERENCES

Bloom, B. (1970). Space/time trade-offs in hash coding with allowable errors. *CACM*, 13(7):422–426.

Carbone, P. et al. (2015). Apache Flink: Stream and batch processing in a single engine. *IEEE TCDE*, 36(4).

Christen, P. (2012). *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer.

Christen, P., Schnell, R., Vatsalan, D., and Ranbaduge, T. (2017). Efficient cryptanalysis of bloom filters for privacy-preserving record linkage. In *PAKDD*.

Christen, P. and Vatsalan, D. (2013). Flexible and extensible generation and corruption of personal data. In *ACM CIKM*, pages 1165–1168.

Dal Bianco, G., Galante, R., and Heuser, C. A. (2011). A fast approach for parallel deduplication on multicore processors. In *ACM SAC*, pages 1027–1032.

Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *CACM*, 51(1).

Durham, E. A. (2012). *A framework for accurate, efficient private record linkage*. PhD thesis, Vanderbilt University.

Fisher, J., Christen, P., Wang, Q., and Rahm, E. (2015). A clustering-based framework to control block sizes for entity resolution. In *Proc. KDD*.

Forchhammer, B., Papenbrock, T., Stening, T., Viehmeier, S., Draisbach, U., and Naumann, F. (2013). Duplicate Detection on GPUs. In *Proc. BTW*.

Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613. ACM.

Karakasidis, A. and Verykios, V. S. (2009). Privacy preserving record linkage using phonetic codes. In *Proc. BCI*.

Karapiperis, D. and Verykios, V. S. (2013). A distributed framework for scaling up LSH-based computations in privacy preserving record linkage. In *Proc. BCI*.

Karapiperis, D. and Verykios, V. S. (2014). A distributed near-optimal LSH-based framework for privacy-preserving record linkage. *ComSIS*, 11(2):745–763.

Karapiperis, D. and Verykios, V. S. (2015). An LSH-based blocking approach with a homomorphic matching technique for privacy-preserving record linkage. *IEEE TKDE*, 27(4):909–921.

Karapiperis, D. and Verykios, V. S. (2016). A fast and efficient hamming LSH-based scheme for accurate linkage. *KAIS*, 49(3):861–884.

Kolb, L., Thor, A., and Rahm, E. (2012). Dedoop: Efficient Deduplication with Hadoop. *PVLDB*, 5(12).

Kolb, L., Thor, A., and Rahm, E. (2013). Don't match twice: Redundancy-free similarity computation with MapReduce. In *DanaC*, pages 1–5. ACM.

Köpcke, H. and Rahm, E. (2010). Frameworks for entity matching: A comparison. *DKE*, 69(2):197–210.

Kroll, M. and Steinmetzer, S. (2014). Automated cryptanalysis of bloom filter encryptions of health records. *ICHI*.

Kuzu, M., Kantarcioglu, M., Durham, E., and Malin, B. (2011). A constraint satisfaction cryptanalysis of bloom filters in private record linkage. In *Proc. PETS*.

Kuzu, M., Kantarcioglu, M., Durham, E. A., Toth, C., and Malin, B. (2013). A practical approach to achieve private medical record linkage in light of public resources. *JMIA*, 20(2).

Mitzenmacher, M. and Upfal, E. (2005). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

Ngomo, A.-C. N., Kolb, L., Heino, N., Hartung, M., Auer, S., and Rahm, E. (2013). When to reach for the cloud: Using parallel hardware for link discovery. In *ESWC*, pages 275–289. Springer.

Niedermeyer, F., Steinmetzer, S., Kroll, M., and Schnell, R. (2014). Cryptanalysis of basic bloom filters used for privacy preserving record linkage. *JPC*, 6(2):59–79.

Odell, M. and Russell, R. (1918). The soundex coding system. *US Patents*, 1261167.

Schnell, R. (2015). Privacy-preserving record linkage. *Methodological Developments in Data Linkage*, pages 201–225.

Schnell, R., Bachteler, T., and Reiher, J. (2009). Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making*, 9(1):41.

Schnell, R., Bachteler, T., and Reiher, J. (2011). A novel error-tolerant anonymous linking code. *German Record Linkage Center, No. WP-GRLC-2011-02*.

Schnell, R. and Borgs, C. (2016). Randomized response and balanced bloom filters for privacy preserving record linkage. In *ICDMW*, pages 218–224. IEEE.

Sehili, Z., Kolb, L., Borgs, C., Schnell, R., and Rahm, E. (2015). Privacy preserving record linkage with PPJoin. In *Proc. BTW*.

Vatsalan, D., Christen, P., O'Keefe, C. M., and Verykios, V. S. (2014). An evaluation framework for privacy-preserving record linkage. *JPC*, 6(1):3.

Vatsalan, D., Christen, P., and Verykios, V. S. (2013). A taxonomy of privacy-preserving record linkage techniques. *Information Systems*, 38(6):946–969.

Vatsalan, D., Sehili, Z., Christen, P., and Rahm, E. (2017). Privacy-preserving record linkage for Big Data: Current approaches and research challenges. *Handbook of Big Data Technologies*.

Wang, C. et al. (2010). MapDupReducer: Detecting near duplicates over massiv datasets. In *Proc. SIGMOD*.