Knowl Inf Syst (2016) 49:861–884
DOI 10.1007/s10115-016-0919-y

REGULAR PAPER

A fast and efficient Hamming LSH-based scheme for accurate linkage

Dimitrios Karapiperis[1] · **Vassilios S. Verykios**[1]

Received: 13 October 2014 / Revised: 3 December 2015 / Accepted: 20 January 2016 /
Published online: 3 February 2016

Abstract In this paper, we propose an efficient scheme for privacy-preserving record linkage by using the Hamming locality-sensitive hashing technique as the blocking mechanism and the Bloom filter-based encoding method for anonymizing the data sets at hand. We achieve highly accurate results and simultaneously reduce significantly the computational cost by minimizing the number of distance computations performed. Our scheme provides theoretical guarantees for identifying the similar anonymized record pairs by conducting redundant blocking and by performing a distance computation only if the corresponding anonymized record pair is formulated a specified number of times. A series of experiments illustrate the efficacy of our scheme in identifying the similar record pairs, while simultaneously keeping the running time exceptionally low.

Keywords Privacy-preserving record linkage · Locality-sensitive hashing · Blocking · Bloom filters

1 Introduction

The process of linking records, which lack global unique identifier, from disparate data sets in order to identify those pairs of records that refer to the same real-world entity, is known as the record linkage, entity resolution, and data matching problem [6]. When data to be matched are deemed to be sensitive or private, such as health data or data kept by national security agencies, specialized privacy-preserving techniques should be applied, which allow data integration and analysis. Privacy-preserving record linkage (PPRL) [15] investigates how to make linkage computations securely by respecting the privacy of the data. For this reason, input records undergo a necessary anonymization process which embeds those records into an anonymization space, where the underlying data are kept private.

✉ Dimitrios Karapiperis
dkarapiperis@eap.gr

1 School of Science and Technology, Hellenic Open University, Patras, Greece

PPRL consists of two steps. During the first step, potentially matching pairs of records are formulated, while during the second step, these pairs are matched. The first step is commonly termed as the *blocking* step, which should be as *efficient* as possible given large data sets. By the term efficient, we mean that we should perform the least possible distance computations in order to identify the matching pairs, regardless of the size of the data sets at hand. The second step, known as the *matching* step, performs the distance computations between the pairs which have been brought together for comparison during the previous step. The distance computations are performed in an approximate manner, which is a core component of PPRL, since the field values of records owned by different data custodians and indeed refer to the same real-world entity, usually contain variations, errors, misspellings, and typos. An illustrative example might be the submission of anonymized medical records, which lack a common unique identifier, by hospitals to an independent trusted party, which in turn identifies any potentially matching pairs by processing the anonymized records.

The blocking solutions, which have been developed to solve the PPRL problem, rely mostly either on clustering [19,26,36] or on tree-based [18,32] techniques. The efficiency of the clustering-based techniques is highly dependent on the public reference set of values, which is used by the participating parties in order to generate clusters of similar records independently. On the other hand, the tree-based indexing methods, as reported and proved in [1,14,38], degrade rapidly even with moderate dimensionality ($\geq$10), because during the matching step, nearly all indexed records are accessed.

In [9,20,24], redundant randomized methods are used, where each record is blocked to several independent blocking groups, which amplify the probability of bringing together similar records for comparison. These methods perform by far better than the traditional methods given large data sets, including high-dimensional records, by exploiting the efficiency of the locality-sensitive hashing (LSH) technique [13]. The authors though utilize an arbitrary number of blocking groups which may either perform unnecessary and expensive distance computations or miss matching record pairs. Recently, the authors in [22] proposed the BfH method which quantifies the absolutely necessary number of blocking groups that are required in order to provide theoretical guarantees in identifying the similar anonymized pairs. Despite these guarantees, given large data sets, the number of pairs which are formulated for comparison is also large and consequently consumes running time at high levels. Specifically, in [22], the number of the matching pairs remains below 10 % of the total number of the formulated pairs.

In this paper, we propose an efficient scheme, namely the frequent pairs scheme (FPS), which improves significantly the running time during the linkage process without sacrificing the accuracy. Our scheme is applied on the string values of the data sets at hand, which are first embedded into the Hamming metric space by using the Bloom filter-based encoding method [33]. Then, these Bloom filters generated are blocked several times into independent blocking groups by applying the locality-sensitive hashing technique. The theoretical premise behind our scheme is the number of times, termed as *collisions*, a certain Bloom filter pair is formulated during the blocking step. We formally prove that the number of collisions a Bloom filter pair achieves, reflects its Hamming distance. We quantify the exact number of collisions that those pairs should exhibit in order to be considered for comparison during the matching step. By doing so, we exclude a large number of pairs and achieve great savings in terms of running time. We also quantify the exact number of blocking groups that are required toward providing theoretical guarantees for identifying each similar pair with a specified confidence. The experimental results in Sect. 6 confirm the efficiency of our scheme, where the number of the matching pairs may be up to 90 % of the total number of the formulated pairs. We also compare with four state-of-the-art-methods which are: (a) the BfH method, (b) the h-CC algorithm of the HARRA suite [24], (c) the multi-dimensional privacy-preserving blocking (MPPB) method [19], and (d) the embedding scheme of strings in the Euclidean space presented in [32].

The structure of the paper is organized as follows: In Sect. 2, the related work is presented. Problem is formally defined in Sect. 3, while in Sect. 4 we present the elementary components of FPS, namely the anonymization process, and the redundant blocking technique. In Sect. 5, we illustrate the theoretical ground of FPS, which is evaluated and compared to other state-of-the-art methods in Sect. 6. Conclusions and ideas for future extensions are discussed in Sect. 7.

2 Related work

In the literature, several privacy-preserving blocking solutions have been proposed such as [9,17–20,22,32,35,36]. A method that had great impact on the research community is the sorted neighborhood approach [16], including all its privacy-preserving variants, e.g., [19,36], which sorts all records from the participating data sets, and then uses a fixed-sized sliding widow over the sorted records in order to compare record pairs which have been formulated within that window. Kuzu et al. [26] proposed a private blocking approach which is based on the differential privacy paradigm [11]. In this approach, a trusted third party, by using public reference values, creates global clusters into which each data custodian independently inserts her records. Then, these records are compared using secure multi-party (SMC) computations [29]. However, methods which rely on the use of a common set of reference values, e.g., those presented in [19,30,36] attain accurate results only when this reference set is a superset of the values included in the data sets at hand.

The two-party techniques developed in [35,36,39] although eliminating the risk of collusion among the participating parties, incur high computation, and communication cost. For instance, Vatsalan et al. [36] introduced a two-party clustering technique where each cluster is generated by using a common set of reference values and additionally contains at least *k* records for providing enhanced privacy guarantees. Nevertheless, the data custodians need to exchange the reference values of each generated cluster, which should be merged and sorted, in order to identify any common clusters and formulate record pairs. Then, these pairs should be compared using a computationally expensive SMC technique.

Recently, the randomized blocking methods, e.g., [9,20,22], have received much attention, since they both provide theoretical accuracy guarantees in the anonymization space and also handle the large data sets efficiently. We refer the interested reader to two recent surveys presented in [5,37].

The Bloom filter-based encoding methods, presented in [10,33,34] and used by our proposed scheme, are easily implemented and preserve the distances of the original record pairs. However, under certain circumstances, these methods are susceptible to cryptanalysis [25,28].

Scannapieco et al. [32] presented a scheme for embedding strings into a Euclidean space in a private manner, by utilizing a trusted third party. We use this scheme as one of our competitors and evaluate it in Sect. 6. Yakout et al. [39], by relying on [32], introduced a two-party approach which maps the records to points in the complex plane and then performs distance computations only for those points which lie within a predefined proximity threshold. Bonomi et al. presented in [3] another embedding scheme which instead of relying on random string values for building the reference sets, it uses the frequent Q-grams extracted from the data sets at hand by performing a time-consuming mining phase.

3 Problem definition

Let us suppose a number of data custodians who engage themselves into a process for identifying the common entities among their records. Without loss of generality, we assume two data custodians who own the data sets A and B, respectively. The data custodians first have to agree on a common schema (a set of common fields) based on which they can actually compare their records. They also need to provide a third field, let us call it *Id*, which plays the role of uniquely identifying each record. In order to accomplish their goals, the data custodians should make use of the services offered by an independent party whom we call G. However, the records owned by the data custodians are deemed private; therefore, they cannot be submitted to G without first undergoing an anonymization process. Toward this end, the data custodians embed the strings of the respective fields of their records into an anonymization space. We denote by A' ($n_{A'} = |A'|$) and B' ($n_{B'} = |B'|$) the anonymized versions of A and B, respectively. We assume that G follows the Honest-but-Curious (HBC) model [37], in that she follows the protocol steps while being curious to learn about other party's data. We also assume that there will be no collusion among the participating parties.

Definition 3.1 *(A similar anonymized record pair)* An anonymized record pair is considered as similar if the distance of the constituent anonymized records is below or equal to a threshold ϑ specified in the used anonymization space. It follows that the corresponding original record pair is also similar if the used anonymization space preserves the distances from the original space of strings.

Each record in a secure anonymized form is submitted to G who utilizes an efficient mechanism in order to identify the set of the similar anonymized record pairs, denoted by $\mathcal{M}$, as fast as possible. Let us also denote the set M which consists of the truly matching pairs, which refer to the same real-world entity, that are also similar in the original space of strings.

Definition 3.2 *(Problem definition)* Given the large volume of records in A' and B', G should produce set M, which should coincide with set $\mathcal{M}$ to the highest possible extent, by consuming the least possible running time.

To this end, the distance computations should be performed between similar anonymized record pairs. A motivating example might be the submission of anonymized personal data of customers of financial institutions on an ongoing basis to a central financial authority. Next, this authority constructs the set $\mathcal{M}$ by using the anonymized data submitted. A pair in $\mathcal{M}$ may potentially correspond to a single customer who, e.g., maintains multiple accounts at several banks worldwide. By combining such pieces of voluminous data very fast, one may make important inferences with respect to the financial status of certain customers. This example becomes more challenging as personal data of these customers may contain variations, errors, and typos. To this end, the mechanism used by G should be as efficient as possible in performing approximate matching operations between anonymized records.

4 Background

In this section, we give a brief outline of the elementary components utilized by our proposed scheme. We list in Table 1 the main variables used throughout this paper.

Table 1 Interpretation of the most important variables used in this paper

T_l	The lth blocking group (hash table) where $l = 1, \dots, L_{opt}$
H_l	A composite hash function used to block a Bloom filter to some T_l
K	The number of the base hash functions used in an H_l
S	The size of a record-level Bloom filter
d_H	The Hamming distance between a pair of Bloom filters
δ	The confidence parameter which bounds above the probability of failure for the similar Bloom filter pairs during the blocking mechanism
ϑ	The distance threshold
p_ϑ^K	The probability of collision for Bloom filter pairs whose distance is equal to ϑ
C	The number of collisions achieved by a Bloom filter pair
$\mathcal{C}$	The number of collisions that a Bloom filter pair should achieve in order to be considered as a frequent pair
$L_\mathcal{C}$	The number of blocking groups utilized by FPS
$\mathcal{F}$	The set of frequent pairs

4.1 Anonymization of records using Bloom filters

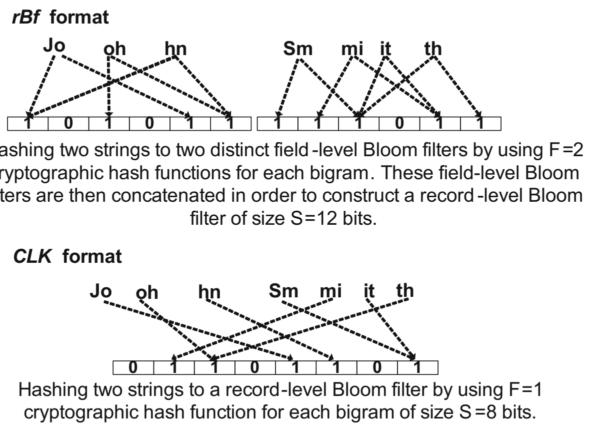
A Bloom filter is a data structure used to represent the elements of a set in order to support membership queries for these elements efficiently, in terms of time and space required. More specifically, the authors in [33] use bitmap arrays of size S, initialized with zeros, where certain components are set to 1 by hashing the bigrams[1] of the strings of a record. Each bigram x is hashed by using F independent composite cryptographic hash functions of the form $\mathcal{G}_i(x)_i = [g_1(x) + (i \;\, g_2(x))] \mod S$, where $i = 0, \dots, F - 1$, and both g_1 and g_2 are keyed hash message authentication code (*HMAC*) functions like *HMAC-MD5* and *HMAC-SHA2*,[2] which utilize a secret key that should be shared only by the data custodians. It has been demonstrated in [9,22,33] that by embedding strings into the Bloom filter (Hamming) space $\mathcal{S} = \{0, 1\}^S$, distances between the original strings are preserved.

Each record is encoded into a record-level Bloom filter by using two formats. The first format, termed as *rBf* [33], uses concatenated field-level Bloom filters, whose size is 500 bits. Each such Bloom filter represents a string of a field, whose bigrams are hashed by $F = 15$ hash functions. The other format, termed as *CLK* [34], uses a single Bloom filter of a larger size, e.g., $S = 1{,}000$, for hashing the bigrams of all the strings of a record, by using $F = 10$ hash functions. Throughout the rest of the manuscript, when referring to a Bloom filter, we denote a record-level Bloom filter. These two Bloom filter formats are demonstrated in Fig. 1.

The cryptographic hash functions, the secret key, as well as the order of concatenation are only known to the data custodians. Should these pieces of information leak, an adversary could establish a frequency attack by using these Bloom filters and publicly available resources such as voter lists, or telephone catalogs.

[1] A bigram is pair of adjacent characters in a string.

[2] http://tools.ietf.org/html/rfc2104.



Hashing two strings to two distinct field-level Bloom filters by using F=2 cryptographic hash functions for each bigram. These field-level Bloom filters are then concatenated in order to construct a record-level Bloom filter of size S=12 bits.

Hashing two strings to a record-level Bloom filter by using F=1 cryptographic hash function for each bigram of size S=8 bits.

Fig. 1 Creating record-level Bloom filters by using both the *rBf*, and *CLK* format

4.2 The Hamming LSH-based blocking technique

FPS uses the Hamming LSH-based blocking technique [9,22] in order to block the produced Bloom filters and formulate pairs for comparison. The principal component of this technique is a bunch of L independent hash tables, termed also as blocking groups. Each hash table, denoted by T_l, where $l = 1, \dots, L$, consists of key-bucket pairs. A bucket hosts a linked list of *Id*'s which belong to potentially similar Bloom filters. The reasoning behind this grouping will be specified later. Additionally, each hash table has been assigned a composite hash function H_l which consists of a fixed number, say K of base hash functions $h_l^{(k)}$ concatenated, where $k = 1, \dots, K$. A base hash function applied to a Bloom filter returns the value of its sth component where $s \in \{0, \dots, S-1\}$ chosen uniformly at random.

Definition 4.1 *(A locality-sensitive hash family)* A family $\mathcal{H}$ of base hash functions has the following key property for any $Bf_1, Bf_2 \in \mathcal{S}$ [13]:

$$\textit{If } d_H \leq \vartheta \textit{ then } \Pr[h_l^{(k)}(Bf_1) = h_l^{(k)}(Bf_2)] \geq p_\vartheta, \tag{1}$$

where $p_\vartheta = 1 - \frac{\vartheta}{S}$ and d_H denotes the Hamming distance between Bf_1 and Bf_2.

The Hamming distance between two Bloom filters is equal to the number of components in which these Bloom filters exhibit different bits. For the sake of simplicity, throughout the rest of the manuscript by using the term distance we refer to the Hamming distance. Intuitively, the smaller the distance is, the higher the probability of a base hash function of producing the same result. The result of an H_l applied to a Bloom filter specifies to which bucket, termed also as block, of a T_l the *Id* of that Bloom filter will be stored. Figure 2 illustrates how two similar Bloom filters are hashed to some T_l.

This *redundant* blocking model amplifies the probability of collision for similar Bloom filters but also increases the running time and the utilized space. Therefore, the optimal number of blocking groups, denoted by L_{opt}, should be used so that the set of the formulated pairs should include as many as possible from the similar pairs. Thus, by setting [2]:

$$L_{opt} = \left\lceil \frac{\ln(\delta)}{\ln(1 - p_\vartheta^K)} \right\rceil \tag{2}$$

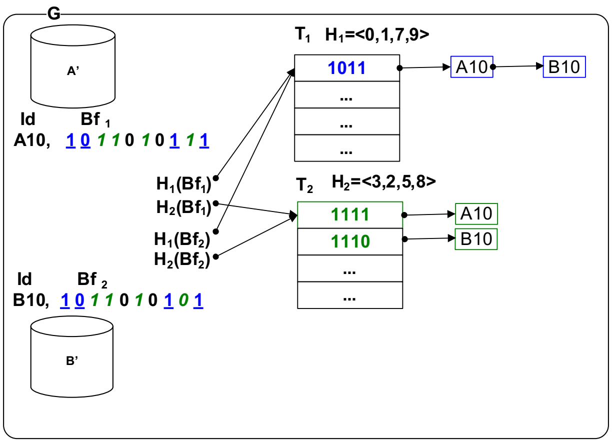


Fig. 2 Snapshot of the blocking mechanism using $L_{opt} = 2$ and $K = 4$. The bits chosen by the hash functions H_1 and H_2 are *underlined* and *italicized*, respectively

each similar Bloom filter pair is returned with high probability at least $1 - \delta$, since the confidence parameter δ, which specifies the probability of failing to return such pairs, is usually set to a small value, e.g., $\delta = 0.1$.

The value for K can be set empirically since the correctness of the scheme is guaranteed by setting appropriately L_{opt} but by doing so, we may not be optimal in terms of running time. In [21], a method for choosing the optimal value is presented, with respect to the Min-Hash LSH family [31], which can be easily adapted to the Hamming family. This method relies on Bloom filter pairs sampling, where by providing several values for K, the one that minimizes the running time is finally chosen. This value, regardless the way we choose it, should be sufficiently large because otherwise a small number of buckets is generated in each T_l, which are overpopulated by Bloom filters resulting in the formulation of mostly dissimilar pairs. This happens because the results of the H_l's, which map the Bloom filters to buckets, do not reflect sufficiently the variations of the bit sequences of those Bloom filters.

5 The frequent pairs scheme

In this section, we present our proposed scheme FPS, used as a shorthand for '*Frequent Pairs Scheme*.' FPS follows the *redundant* blocking model of the Hamming LSH-based blocking, as described before, where a certain pair of Bloom filters might be formulated in several T_l's exhibiting a number of *collisions*.

Definition 5.1 *(A collision)* A collision is a Bloom filter pair formulated in some T_l. The number of collisions for a certain Bloom filter pair is denoted by C where $C \in \{1, \ldots, L_{opt}\}$.

The novel notion of collisions is the main theoretical premise of FPS, which correlates the number of collisions[3] a certain Bloom filter pair achieves with its distance. Our scheme first identifies these pairs that exhibit a certain number of collisions, which will be exactly specified later, and then performs the distance computations only for these pairs.

[3] LSH-based collision counting has also been studied in [12], but the underlying theoretical foundations and the techniques used therein are completely different than ours.

During the blocking step, certain Bloom filter pairs are being blocked together in multiple T_l's, which might be an indication of similarity; especially, the pairs which exhibit C collisions where $C = L_{opt}$, should be similar with overwhelming probability. Intuitively, the more the collisions are, the higher the probability of similarity is.

We next specify the required number of collisions in terms of a threshold, denoted by $\mathcal{C}$, that a Bloom filter pair should achieve in order to be considered as a potential matching pair. Let C_ϑ be the random variable that counts the number of collisions of any Bloom filter pair (Bf_1, Bf_2) whose distance is exactly equal to ϑ. Since each H_l consists of K base hash functions, the probability of collision in some T_l for these pairs is p_ϑ^K. Hence, C_ϑ is binomially distributed as:

$$C_\vartheta \sim bin(L_{opt}, p_\vartheta^K), \tag{3}$$

where $bin(\cdot, \cdot)$ denotes the binomial distribution, L_{opt} is the number of trials, and p_ϑ^K is the probability of success.

Lemma 5.1 *The expected number of collisions $E[C_\vartheta]$ of any Bloom filter pair (Bf_1, Bf_2) whose distance is equal to ϑ is:*

$$E[C_\vartheta] = L_{opt}\, p_\vartheta^K. \tag{4}$$

Proof

$$E[C_\vartheta] = E\left[\sum_{l=1}^{L_{opt}} I_l^{(\vartheta)}\right] = \sum_{l=1}^{L_{opt}} E[I_l^{(\vartheta)}] = L_{opt}\, p_\vartheta^K, \tag{5}$$

where $I_l^{(\vartheta)}$ indicates a collision of (Bf_1, Bf_2) in some T_l, defined as:

$$I_l^{(\vartheta)}(Bf_1, Bf_2) = \begin{cases} 1, & \text{if} \quad H_l(Bf_1) = H_l(Bf_2) \\ 0, & \text{if} \quad H_l(Bf_1) \neq H_l(Bf_2) \end{cases}$$

and $E[I_l^{(\vartheta)}] = p_\vartheta^K$. If $d_H < \vartheta$, then the probability of collision is bounded from below by p_ϑ^K since p_ϑ^K increases as ϑ decreases for a fixed K. □

Random variable C_ϑ achieves the number of collisions shown in Lemma 5.1 *on expectation* with standard deviation $\sigma = \sqrt{L_{opt}\ \ p_\vartheta^K\ (1 - p_\vartheta^K)}$.

Corollary 5.1 *The expected number of collisions $E[C_\vartheta]$ depends on the confidence parameter δ.*

Proof We expand Eq. (4) from Lemma 5.1, which is written as:

$$E[C_\vartheta] = L_{opt}\, p_\vartheta^K = \frac{\ln(\delta) p_\vartheta^K}{\ln(1 - p_\vartheta^K)} \approx \frac{\ln(\delta) p_\vartheta^K}{-p_\vartheta^K} = \ln(\frac{1}{\delta}), \tag{6}$$

where we use the fact that $(1 - p_\vartheta^K) \leq e^{-p_\vartheta^K}$ and substitute $\ln(1 - p_\vartheta^K)$ with $\ln(e^{-p_\vartheta^K})$. □

Corollary 5.2 *Given any Bloom filter pair, with distance τ where $\tau \leq \vartheta$, that exhibits a number of collisions C_τ, it follows from Lemma 5.1 that*

$$E[C_\tau] \geq E[C_\vartheta]. \tag{7}$$

Proof

$$p_\tau^K \geq p_\vartheta^K \Rightarrow E[C_\tau] \geq E[C_\vartheta].$$

□

Table 2 Indicative configuration parameters for FPS

K	30
L_{opt}	14
p_{ϑ}^{K}	0.293
δ	0.01
ϑ	80
S	2000
$\mathcal{C}$	2
$L_{\mathcal{C}}$	20

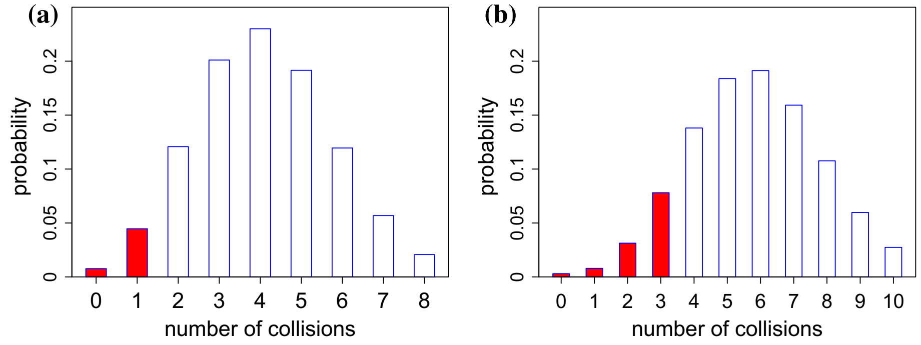


Fig. 3 The *colored bars* indicate the distribution of probability for pairs whose distance is equal to ϑ of exhibiting C collisions where $C \in \{0, \ldots, \mathcal{C} - 1\}$

Thus, $E[C_{\vartheta}]$ is the expected lower bound of collisions for the similar pairs.

Definition 5.2 *(A frequent pair)* A Bloom filter pair which achieves a number of collisions $C \geq \mathcal{C}$ where

$$\mathcal{C} = \| E[C_{\vartheta}] - \sigma \|$$

is a frequent pair and is also expected to be a similar pair. We symbolize by $\| \cdot \|$ the nearest integer value.

The set of the frequent pairs is denoted by $\mathcal{F}$. We subtract one standard deviation σ from $E[C_{\vartheta}]$ in order to identify those similar pairs whose number of collisions is slightly below $\mathcal{C}$. As a result of doing this, we minimize the accuracy loss. In the next subsection, we quantify the accuracy loss of FPS with a specified probability.

5.1 Specifying the level of confidence

By using the binomial distribution with the parameters shown in (3), the cumulative probability of any Bloom filter with $d_H = \vartheta$ exhibiting less than $\mathcal{C}$ collisions bounds from above the accuracy loss of FPS.

By using the parameters in Table 2, which yield $\mathcal{C} = 2$, the value of the cumulative distribution function (cdf) is $\Pr[C < 2] \simeq 0.0523$. Thus, the probability of missing pairs with $d_H \leq \vartheta$ is bounded above by 0.0523.[4] By setting $\delta = 0.001$, we obtain $\mathcal{C} = 4$ and $\Pr[C < 4] \simeq 0.1181$. These probability distributions are illustrated in Fig. 3a, b, respectively.

[4] The cumulative probability for pairs with $d_H < \vartheta$ is less than 0.0523 due to the higher success probability yielded by the smaller distances than ϑ.

Therefore, by setting smaller values to δ, we obtain larger values both for $\mathcal{C}$, as Corollary 5.1 suggests, and for the cdf $\Pr[C < \mathcal{C}]$. As we notice though, these values of the cdf are quite larger than the corresponding values of δ, which implies potential loss of accuracy. For this reason, we increase the number L_{opt} of blocking groups so that the similar pairs will have the chance to collide more frequently. The new value for L_{opt}, that is $L_{\mathcal{C}}$, should be adjusted to the values of $\mathcal{C}$ and δ.

Theorem 5.1 *Each similar pair is included in $\mathcal{F}$ by using $L_{\mathcal{C}}$ blocking groups, where*

$$L_{opt} < L_{\mathcal{C}} < \frac{(\mathcal{C}-1) - \ln(\delta) + \sqrt{(\ln(\delta))^2 - 2(\mathcal{C}-1)\ln(\delta)}}{p_{\vartheta}^{K}}, \tag{8}$$

with confidence at least $1-\delta$.

Proof A straightforward way to specify $L_{\mathcal{C}}$ is by bounding the cdf $\Pr[C < \mathcal{C}]$ using the following Chernoff inequality [27]:

$$\Pr[C < \mathcal{C}] \leq \exp\left\{-\frac{(L_{\mathcal{C}}\, p_{\vartheta}^{K} - (\mathcal{C}-1))^2}{2\, L_{\mathcal{C}} p_{\vartheta}^{K}}\right\}, \tag{9}$$

for the binomial distribution $bin(L_{\mathcal{C}}, p_{\vartheta}^{K})$, where $\mathcal{C} \leq L_{\mathcal{C}}\, p_{\vartheta}^{K}$. We solve for $L_{\mathcal{C}}$ and obtain:

$$L_{\mathcal{C}} = \frac{(\mathcal{C}-1) - \ln(\delta) + \sqrt{(\ln(\delta))^2 - 2(\mathcal{C}-1)\ln(\delta)}}{p_{\vartheta}^{K}}. \tag{10}$$

(see the solution in the "Appendix")

This bound though is not tight and imposes unnecessary running time and space. For instance, by using the parameters in Table 2, the value of $L_{\mathcal{C}}$ increases considerably to 38. For this reason, we apply a different strategy by iteratively increasing L_{opt} by 1 and by calculating the value of the cdf $\Pr[C < \mathcal{C}]$, which refers to a new binomial distribution using the newly derived value of L_{opt}. We perform these iterations until $\Pr[C < \mathcal{C}] < \delta$. □

By doing so, in the previous example, we obtain $L_{\mathcal{C}} = 20$ blocking groups.

As distances increase beyond ϑ, the probability of the corresponding pairs to achieve additional collisions decreases, as well as to achieve less collisions than $\mathcal{C}$ increases, which is quite convenient for our scheme. This behavior is clearly illustrated in Fig. 4, where for $\mathcal{C} = 2$, the value of the cdf $\Pr[C \geq \mathcal{C}]$ for similar pairs is much higher than for pairs with distances greater than ϑ. In the same sense, the value of the cdf $\Pr[C < \mathcal{C}]$ increases, as the distances increase.

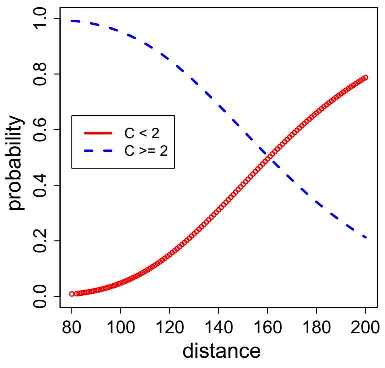


Fig. 4 By setting $\delta = 0.01$ and $\mathcal{C} = 2$, as distances increase, the value of $\Pr[C < \mathcal{C}]$ increases, while the value of $\Pr[C \geq \mathcal{C}]$ decreases

Algorithm 1 Counting the collisions of a $Bf_{B'}$ in the buckets of the T_l's.

```
Require: $Bf_{B'} \in B'$
1: $\mathcal{U} \leftarrow$ new $UniqueCollection()$    ▷ $\mathcal{U}$ is a collection of unique $Id$s.
2: for $l = 1, \ldots, L_{\mathcal{C}}$ do
3:     $Id\_list \leftarrow T_l.get(H_l(Bf_{B'}))$    ▷ Object $Id\_list$ is a linked list of $Id$s.
4:     for $i = 1, \ldots, Id\_list.size()$ do
5:         $Id \leftarrow Id\_list[i]$
6:         if ($\mathcal{U}.contains(Id)$) then
7:             $\mathcal{U}.inc(Id)$    ▷ Method $inc(\cdot)$ increases by 1 the count of a certain $Id$ stored in $\mathcal{U}$.
8:         else
9:             $\mathcal{U}.add(Id)$    ▷ Method $add(\cdot)$ inserts an $Id$ into $\mathcal{U}$ and sets its count to 1.
10:        end if
11:        $C \leftarrow \mathcal{U}[i].count$
12:        if ( $C = \mathcal{C}$) then
13:            $f\_list$.add($Id$)    ▷ Object $f\_list$ is a linked list which represents set $\mathcal{F}$.
14:        end if
15:    end for
16: end for
17: for $i = 1, \ldots, f\_list.size()$ do
18:     $Id \leftarrow f\_list[i]$
19:     $Bf_{B'} \leftarrow retrieve(Id)$    ▷ Method $retrieve(\cdot)$ retrieves a Bloom filter from a data store.
20:     $d_H \leftarrow distance(Bf_{A'}, Bf_{B'})$
21:     if ($d_H \leq \vartheta$) then
22:         $\mathcal{M}.add(Bf_{A'}, Bf_{B'})$    ▷ Method $add(\cdot)$ adds a similar Bloom filter pair to $\mathcal{M}$.
23:     end if
24: end for
```

5.2 Outline of FPS

Let us denote by $Bf_{A'}$ and $Bf_{B'}$ the Bloom filters which belong to data sets A' and B', respectively. G first hashes each $Bf_{A'}$ and stores their Id's in the buckets of the T_l's. Then, G applies FPS for each $Bf_{B'}$ by counting the corresponding collisions, as conveyed by Algorithm 1. A collection of unique elements, denoted ny $\mathcal{U}$, is used to store each $Id \in A'$, whose $Bf_{A'}$ formulates a pair with $Bf_{B'}$, together with the count of collisions. This collection is initialized for each $Bf_{B'}$. For each bucket that a $Bf_{B'}$ maps to, G retrieves the Id's already stored therein (line 3) and then queries them against $\mathcal{U}$.[5] If an Id is found in $\mathcal{U}$, then G increases its count (lines 6 and 7); otherwise, it is inserted into $\mathcal{U}$ (line 9). Then, by checking whether this count is equal to $\mathcal{C}$ (line 12), that Id is added to the linked list f_list, which materializes the set $\mathcal{F}$ (line 13). G next iterates f_list and performs the distance computations between the corresponding Bloom filter pairs, which are added to set $\mathcal{M}$ if the threshold condition is met (lines 20 and 21).

6 Evaluation

We evaluate FPS in terms of (a) the accuracy in identifying the truly matching record pairs and (b) the efficiency in reducing the number of Bloom filter pairs. For the needs of the experiments, we used data sets which were originated by two sources, namely (a) the NCVR list,[6] and (b) the DBLP bibliography database.[7] The fields used for each data set are listed in Table 3. We developed a software prototype which extracts two data sets A and B, of

[5] Collection $\mathcal{U}$ is instantiated by a *HashBag* object, which is contained in the Apache Commons package http://commons.apache.org/, for Java programming language.

[6] ftp://www.app.sboe.state.nc.us/data/.

[7] http://dblp.uni-trier.de/xml/.

Table 3 Fields used in each record

Field	Avg. bigrams
NCVR	
FirstName	5.1
LastName	5.0
Address	20.0
Town	7.2
DBLP	
FirstName	4.8
LastName	6.2
Title	64.8
Year	3.0

user-defined size and perturbation frequency by using the two sources mentioned before. Insert, edit, delete, and transpose operations are used to perturb the values of the fields. The perturbed records are placed in data set B. We apply:

- A light perturbation scheme, termed as Pt_1, which perturbs three, randomly chosen, fields using an insert, an edit, and a delete perturbation operation.
- A heavy scheme, termed as Pt_2, where we apply the above-mentioned operations to the first, second, and fourth field, respectively. Additionally, we apply an insert and a transpose operation to the values of the third field, which exhibit the largest number of bigrams, in both data sets.

We also illustrate the accuracy of our scheme by increasing (a) the number of perturbation operations and (b) the size of the data sets at hand. The frequency of selecting records for perturbation was set to 0.5. The experiments were executed on a dual-core Pentium PC with 40 GB of main memory. The software components were developed using the Java programming language (JDK 1.7).

6.1 Quality measures and configuration parameters

The Pairs Completeness (*PC*), Pairs Quality (*PQ*), and Reduction Ratio (*RR*) metrics [5] are employed to evaluate the efficiency with respect to accuracy of our scheme. The set of the truly matching record pairs is denoted by M and the set of the identified similar pairs by $\mathcal{M}$. The accuracy in identifying the pairs of M is indicated by the *PC* metric, which is equal to $|\mathcal{M} \cap M|/|M|$. The *PQ* metric shows the efficiency in generating mostly matching pairs with respect to the number of the candidate pairs, defined as $PQ = |\mathcal{M} \cap M|/|CR|$, where *CR* is the set of candidate pairs. The *RR* metric indicates the percentage in the reduction in the comparison space, which is equal to $1.0 - |CR|/|A \times B|$, where $A \times B$ denotes the set of all possible record pairs. Throughout the evaluation, we ran each experiment 50 times and plotted the average values of these metrics in the figures shown below.

We embed the strings into the Bloom filter space by using both the *rBf* and *CLK* formats presented in Sect. 4.1. We use Bloom filters of size $S = 2{,}000$ bits[8] for *rBf*, and $S = 1,000$ bits for *CLK*. Distance thresholds for *rBf* are specified according to the observations reported in [22]. More specifically, an insert (or delete), an edit, and a transpose operation changes around 30, 40, and 80 bits, respectively, in the perturbed Bloom filter. Thus, ϑ is set to

[8] Each record includes four fields; therefore, $S = 4 \times 500$.

Table 4 Statistical analysis of distances

			Avg.	SD
NCVR	Pt_1	*rBf*	45	16
		CLK	25	9
	Pt_2	*rBf*	171	13
		CLK	88	11
DBLP	Pt_1	*rBf*	30	16
		CLK	14	6
	Pt_2	*rBf*	138	15
		CLK	61	11

Table 5 Configuration parameters

		ϑ	δ	$\mathcal{C}$	L_{opt}	$L_{\mathcal{C}}$
rBf	Pt_1	100	0.01	2	20	28
			0.005	3	22	40
			0.001	4	29	56
			0.0005	5	32	105
			0.0001	6	39	129
	Pt_2	210	0.01	2	127	183
			0.005	3	146	255
			0.001	4	190	360
			0.0005	5	209	628
			0.0001	6	253	768
CLK	Pt_1	55	0.01	2	23	34
			0.005	3	27	47
			0.001	4	35	67
			0.0005	5	38	123
			0.0001	6	46	151
	Pt_2	110	0.01	2	150	217
			0.005	3	173	303
			0.001	4	225	426
			0.0005	5	247	742
			0.0001	6	300	908

100 and 210[9] for Pt_1 and Pt_2, respectively. For *CLK*, we specified the distance thresholds by perturbing pairs of strings and then computing the corresponding distances in the Bloom filter space. *CLK* uses a smaller number of cryptographic hash functions than *rBf*, which results in generating pairs with also smaller distances. We empirically quantified these distances which are on average 15,25, and 40 for each perturbation, respectively. Therefore, ϑ is set to 55 for Pt_1 and to 120[10] for Pt_2. Throughout the evaluation, we fix $K = 30$ which combined with these thresholds yields the parameters listed in Table 5.

[9] For Pt_1, we apply an insert, a delete, and an edit operation, thus ϑ for *rBf* should be set to $30+30+40 = 100$ bits, while for Pt_2 to $30 + 30 + 40 + 30 + 80 = 210$ bits due to the two additional operations.

[10] For Pt_1, ϑ for *CLK* should be set to $15+15+25 = 55$ bits, while for Pt_2 to $15+15+25+15+40 = 110$ bits.

Table 4 lists the average distances, as well as the values of the standard deviation, of both Bloom filter formats for each data set used and for each perturbation scheme applied. We notice that the DBLP-based data set exhibits smaller distances than the NCVR-based data set. This behavior is related to the larger number of bigrams contained in the strings of the DBLP-based data set, as conveyed by Table 3. Another key observation is that by applying Pt_2 to the NCVR-based data set, the values of standard deviation show that distances are concentrated very close to ϑ. Therefore, we can evaluate the effectiveness of our scheme in identifying the corresponding pairs. On the other hand, by applying Pt_1, distances lie away from ϑ by using both formats. Therefore, the corresponding pairs are more easily identifiable.

6.2 Baseline methods

We compare FPS with four other state-of-the-art blocking methods, namely BfH [22], EUC [32], MPPB [19], and HARRA [24]. **BfH** is a randomized method that relies on the redundant Hamnming LSH-based blocking, described in Sect. 4.2, and utilizes L_{opt} independent blocking groups. BfH uses Bloom filters, and specifically the *rBf* format, for anonymizing the strings of a record. Each Bloom filter is hashed and inserted into some bucket, of each blocking group, specified by the result of the respective H_l. During the matching step, a *pair* is formulated when a Bloom filter from A' is grouped to the same bucket of some T_l with a Bloom filter from B'. To be more precise, the stored *Id*s in the T_l's are used for retrieving the corresponding Bloom filters from the data stores. These pairs are compared and are classified as matching or as non-matching pairs according to their distance which is compared to a specified threshold ϑ. For BfH, we use the values of thresholds, K, δ, and L_{opt} listed in Table 2.

EUC was proposed by Scannapiecco et al. [32]. This method represents strings in a private manner by embedding them in a Euclidean space. EUC uses P reference sets, common to both data custodians who participate in the linkage process. In these reference sets, each element comprises a random sequence of characters of length approximately equal to the average length of strings in the data sets. Embedding a string s results in a vector of size P, where each component of this vector stores the minimum edit distance of s from all the elements in a reference set. For higher accuracy, we (a) set 30 dimensions for each field and (b) turned off both the greedy re-sampling and the distance approximation heuristic, both illustrated in [32]. The authors, in order to find the similar vectors, use a multi-dimensional tree-based index, which has the performance drawback described in Sect. 2. For this reason, we utilize the Euclidean LSH-based blocking scheme [7,22] specifically developed for finding similar points in Euclidean spaces. For the LSH-based mechanism, we set $K = 5$, and thresholds to 5.5 and 11 for each perturbation scheme, respectively. We experimented with several values for L, and we chose $L = 40$ and $L = 160$ which achieve a good balance between efficiency and accuracy.

MPPB was proposed by Karakasidis et al. [19] and is based on the K-Medoids algorithm [23] for creating clusters from elements of public reference sets. Records are first classified to those clusters and are encoded into Bloom filters. Then, the sorted neighborhood algorithm [16] is used for the formulation of Bloom filter pairs by applying a sliding window within each cluster. MPPB uses the dice coefficient [6] for computing the similarity between the pairs formulated during the previous step. We set similarity thresholds to 0.85 and 0.80 for the two perturbation schemes, respectively.

Kim et al. introduced the h-CC algorithm as part of the **HARRA** suite [24]. HARRA uses the Min-Hash LSH-based technique [4,31] as the blocking mechanism. Hash functions of the Min-Hash family are locality sensitive to the Jaccard distance [31]. Each record is represented as a binary vector, where each component represents a distinct bigram. HARRA sets to 1 those components which refer to bigrams that are contained in the strings of a record. A drawback of this ambiguous approach is that one cannot distinguish to which field a certain bigram belongs. Those vectors also exhibit high degree of sparsity because their size is large in order to accommodate all possible bigrams of a specific alphabet. For example, by considering the uppercase characters, HARRA uses size of 26^2 components for each vector. However, as Table 3 suggests, the number of bigrams for a record is usually small compared to the size of the vectors. During the blocking phase, HARRA randomly permutes the indexes of the components of each vector and chooses the index of the minimum nonzero component, as the value of each base hash function. HARRA performs the blocking and matching steps iteratively for each T_l. When the matching condition is met for a pair, the corresponding records from A and B are removed from the data sets and do not participate in the subsequent iterations. Since HARRA chooses arbitrary values for L, we empirically set $K = 6$ and the distance thresholds to 0.35 ($L = 30$) and 0.45 ($L = 70$) for Pt_1 and Pt_2, respectively.

6.3 Experimental results

For the first set of the experiments, we use the NCVR-based data sets with $n_A = n_B =$ 1,000,000 records.

6.3.1 Confidence parameter δ

We set several values to δ in order to derive different values for the collision threshold $\mathcal{C}$ as illustrated in Table 2. We measured both the *PC* and *PQ* rates using the *rBf*, and *CLK* formats shown in Fig. 5a–d. A general observation is that the *PQ* rates increase almost linearly as the value of δ decreases by using both Bloom filter formats. Figure 5a–c highlights that by applying Pt_1, as the value of δ decreases, where we indirectly require more accuracy, both the *PC* and *PQ* rates achieve remarkable performance. Specifically, the *PQ* rates reach almost 0.93, while the *PC* rates are constantly above 0.95. We note that smaller values of δ entail larger number of blocking groups, which seem not to hamper the efficient formulation of the frequent pairs. By applying Pt_2, the *PC* rates shown in Fig. 5b drop slightly, since distances are tightly concentrated around the specified threshold, as described before. However, FPS achieves to identify most of the similar pairs and only a small amount of them is missed. The larger thresholds for Pt_2 yield larger values for $L_{\mathcal{C}}$, shown in Table 2, which inevitably formulate much more dissimilar pairs than Pt_1. Consequently, by using $\delta = 0.001$, the *PQ* rates are lower, which still remain on a remarkably high level, as can be seen in Fig. 5d. By setting $\delta = 0.0005$ and $\delta = 0.0001$, which entail accuracy of 0.9995 and 0.9999, respectively, FPS required more amount of main memory to build the blocking groups than the one available in our system.

6.3.2 Bloom filter formats

We observe that *rBf* constantly achieves higher *PC* rates than *CLK*, which has the drawback of representing identical bigrams that belong to different fields in the same structure. For instance, *CLK* would consider the pair of strings ‘*Steve Johnson*’ and ‘*John Stevens*’ as similar, although in reality they are quite different. Another drawback of *CLK* is that the results of the cryptographic hash functions overlap much more frequently than *rBf* for different bigrams because a single Bloom filter is used for all the bigrams of a record. This behavior results in the

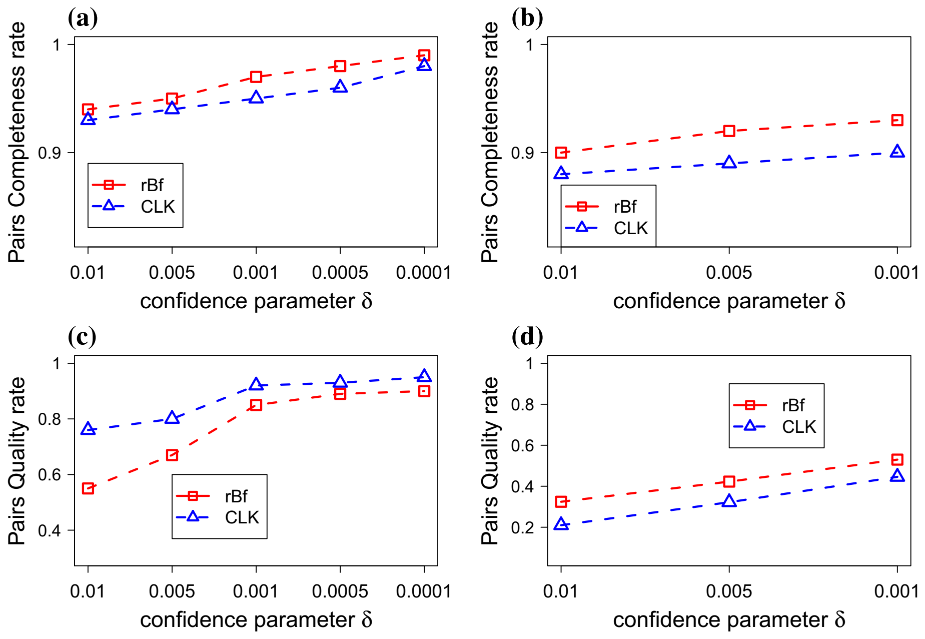


Fig. 5 The *PC* and *PQ* rates achieved by FPS by setting several values to δ. **a** *PC* rates by applying Pt_1. **b** *PC* rates by applying Pt_2. **c** *PQ* rates by applying Pt_1. **d** *PQ* rates by applying Pt_2

obfuscation of the expected distances for certain types of perturbations, which are described in Sect. 6.1. Hence, we may miss some truly matching pairs and misleadingly consider pairs as matching, which in reality they are not. Figure 5c indicates that for Pt_1, the *PQ* rates of *CLK* are much higher than *rBf*. This happens due to the smaller size used by *CLK* for creating the Bloom filters. Thereby, the H_l's have higher probability to reflect the variations of the bit sequences caused by the differing bigrams of the initial similar strings and to produce high-quality buckets in the T_l's, which are populated with mostly similar pairs. In contrast, by applying Pt_2, the larger number of differing bigrams, combined with the smaller size of *CLK*, downgrades its performance. The number of bigrams is another factor that plays an important role in the formulation of the Bloom filters using both formats and consequently in the corresponding distances. For example, by using *rBf* the distance between *'Jones'* and *'Jonas'* is 45. In contrast, the distance between *'Karapiperis'* and *'Karipiperis'* is equal to 28. Comparing the Bloom filters generated by the same strings but by using *CLK*, the distances are 29 and 19, respectively. Therefore, an error between a pair of strings of the *Address* field or of the *Title* field has less impact on the distance between the corresponding Bloom filters than an error between strings of the *LastName* field, which has smaller average number of bigrams as shown in Table 3.

6.3.3 Increasing the number of perturbations

Another interesting view of the results is the performance of FPS in identifying the similar pairs by progressively increasing the number of perturbations applied on the values of a single field by using the *rBf* format. For this series of experiments, we chose the *Address* field whose values contain on average 20 bigrams (21 characters). We initially set threshold

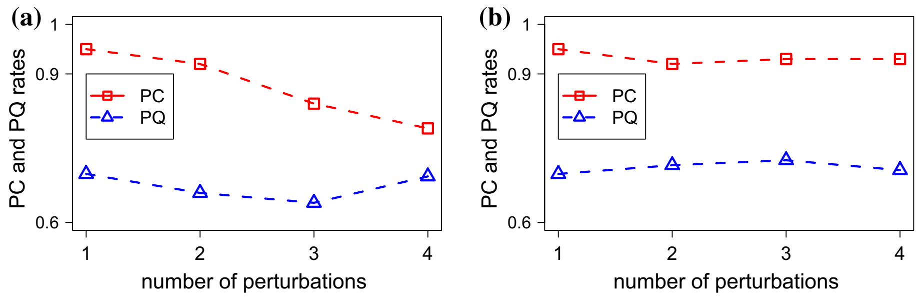


Fig. 6 By applying progressively a certain number of perturbations to the values of a single field, the *PC* rates were restored after increasing the size *S* of the field-level Bloom filter. **a** Size of the Bloom filters of the *Address* field set to $S = 500$ bits. **b** Size of the Bloom filters of the *Address* field set to $S = 550$ bits

ϑ to 180[11] bits, which yielded $L_{\mathcal{C}} = 189$ ($\delta = 0.01$). A natural side effect that follows, which is quite similar to *CLK*'s behavior explained previously, is the increased number of overlapping results when hashing different bigrams during the construction of the Bloom filters. Figure 6a suggests that the *PC* rates were affected negatively as we were increasing the number of perturbations. The remedy was to empirically increase the size *S* of the field-level Bloom filters for the *Address* field by 10 % which yielded $S = 550$ bits and $L_{\mathcal{C}} = 176$ blocking groups. By doing so, the *PC* rates were restored, as Fig. 6b clearly shows, and we also noticed that the *PQ* rates slightly increased because of the smaller number of blocking groups generated due to the higher threshold.

6.3.4 Size *S* of the Bloom filters

We next illustrate the relation of size *S* with the performance of FPS using the *rBf* format, since in *CLK* the size is set to a constant value. We applied an edit operation ($\vartheta = 40$) to the values of the *LastName* field and then gradually increased the number of fields that participated in the linkage process, which resulted in the proportional increase in size *S* without applying any perturbations. Table 6 shows the corresponding *PQ* rates where we observe that as size *S* increases, the *PQ* rates also increase due to the reducing number $L_{\mathcal{C}}$ of the blocking groups that are utilized. We also notice that the *PC* rates remain stable because they are guaranteed by the values of $L_{\mathcal{C}}$. We then perturbed the values of each appended field and increased proportionally the value of ϑ, which yielded a constant value for $L_{\mathcal{C}}$, namely $L_{\mathcal{C}} = 136$. In Table 7, we observe that both the *PC* and *PQ* rates exhibit a very small deviation due to the same number of blocking groups utilized.

6.3.5 Wall-clock time

We complete this set of experiments by measuring the wall-clock time consumed by our scheme given an increasing number of records for both data sets used. Figure 7a, c, for Pt_1, clearly indicates a linear increase in the required time as the number of records increases, where *CLK* is faster than *rBf*. This superiority of *CLK*, in Pt_1, keeps up with the higher *PQ* rates illustrated in Fig. 5c. Not surprisingly, in Pt_2, *rBf* performs slightly faster than *CLK*, as Fig. 7b, d suggest, due to the corresponding higher *PQ* rates achieved, depicted in Fig. 5d.

[11] Since we apply an insert, a delete, an edit, and a transpose operation, ϑ should be set to $30+30+40+80 = 180$ bits.

Table 6 The *PQ* rates increase as the size *S* of the Bloom filters grow and the number of perturbations remains stable ($\mathcal{C} = 2$)

S	ϑ	$L_{\mathcal{C}}$	PQ	PC
500	40	136	0.10	0.98
1000	40	38	0.51	0.98
1500	40	26	0.63	0.99
2000	40	21	0.65	0.98
2500	40	19	0.65	0.99

Table 7 The *PQ* rates remain almost stable as threshold ϑ increases according to the perturbations applied ($\mathcal{C} = 2$)

S	ϑ	$L_{\mathcal{C}}$	PQ	PC
500	40	136	0.12	0.99
1000	80	136	0.11	0.98
1500	120	136	0.12	0.95
2000	140	136	0.14	0.94
2500	180	136	0.10	0.94

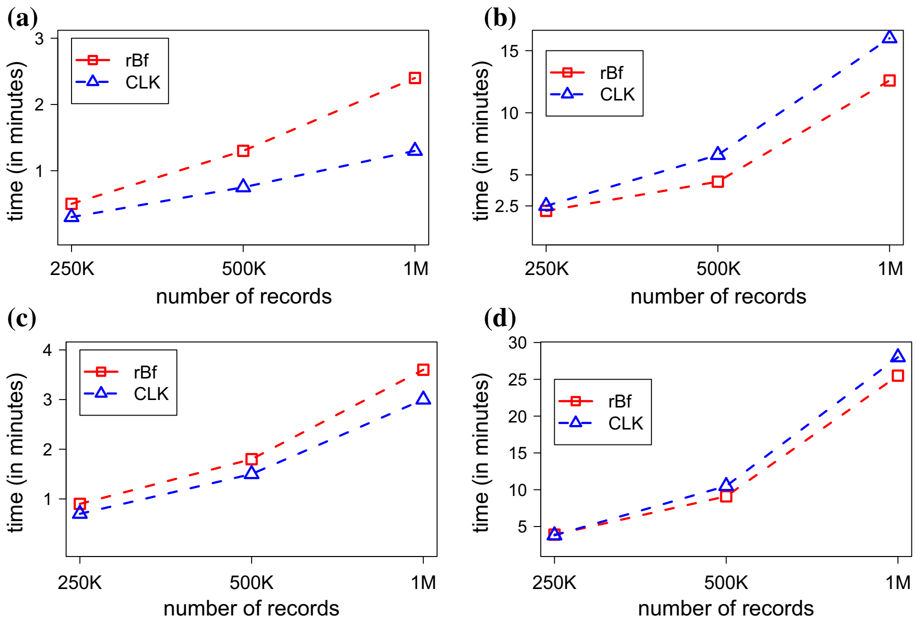


Fig. 7 Measuring wall-clock time by scaling the size of the data sets. **a** Pt_1 and $\delta = 0.01$. **b** Pt_2 and $\delta = 0.01$. **c** Pt_1 and $\delta = 0.001$. **d** Pt_2 and $\delta = 0.001$

6.3.6 Comparing with the baseline methods

The next set of experiments includes the comparison with the baseline methods presented in the previous subsection. Both FPS and BfH use the *rBf* format and δ is set to 0.01. The rest of the parameters are set as above.

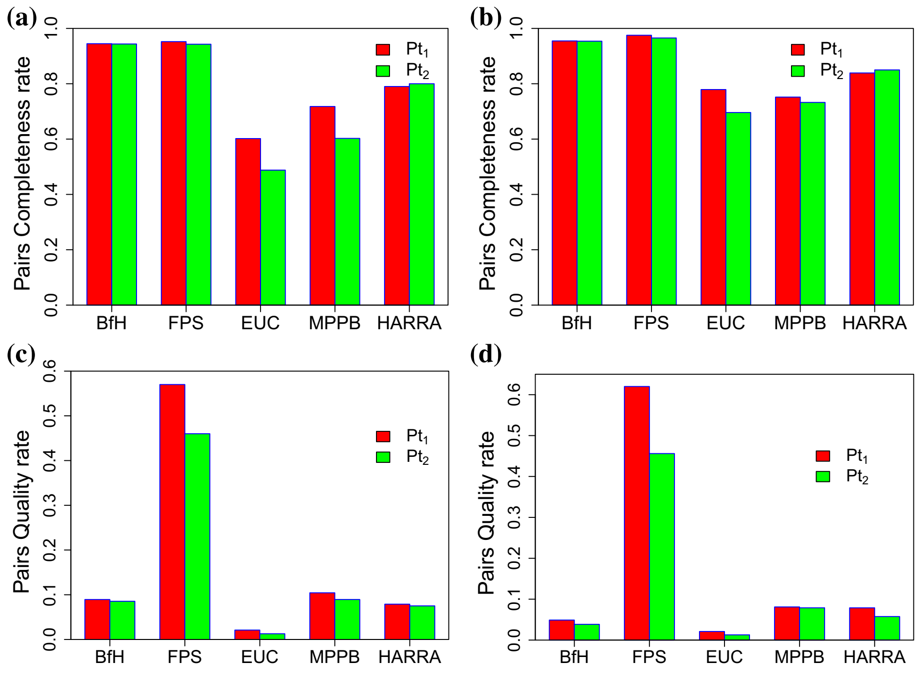


Fig. 8 The *PC* rates of BfH and FPS are constantly above 0.95. Simultaneously, FPS achieves outstanding *PQ* rates. **a** NCVR-based data set. **b** DBLP-based data set. **c** NCVR-based data set. **d** DBLP-based data set

6.3.7 Pairs completeness and pairs quality

Figure 8a, b clearly indicates that both FPS and BfH outperform the other methods by a large margin in terms of accuracy. Both schemes perform better using the DBLP-based data sets due to the smaller distances exhibited, as explained in Sect. 6.1. EUC and MPPB exhibit rather low *PC* rates due to different reasons. EUC falls short in preserving the initial distances of strings, especially in Pt_2, where more perturbations are applied. For MPPB, neither the clustering nor the sliding window steps are capable of formulating similar record pairs. We attempted to set the sliding window size to a large value (>70) in order to improve accuracy, but this affected the performance negatively, since a large number of candidate record pairs were generated. Consequently, the *PQ* rates fell below 0.1, which are slightly better than BfH and by far lower than FPS, as Fig. 8c, d illustrates. These figures highlight the superiority of FPS in formulating mostly similar pairs with outstanding *PQ* rates, which exceed 0.4 for both perturbation schemes. HARRA's accuracy is affected negatively by two factors. The first is the early removal of records of those pairs, which although meet the distance threshold, they do not constitute a truly matching pair. Since those records are then excluded from the remaining iterations, they do not have any other chance to formulate any possible matching pairs. The other factor is the ambiguity caused by the way the bit vectors are created, as described in Sect. 6.2. The *PQ* rates are around 0.1 using both data sets and are almost equal to the rates of FPS and MPPB.

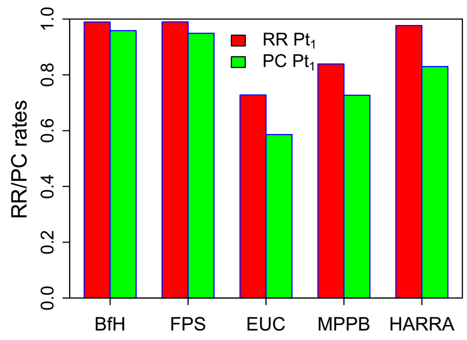


Fig. 9 The *RR* along with the *PC* rates

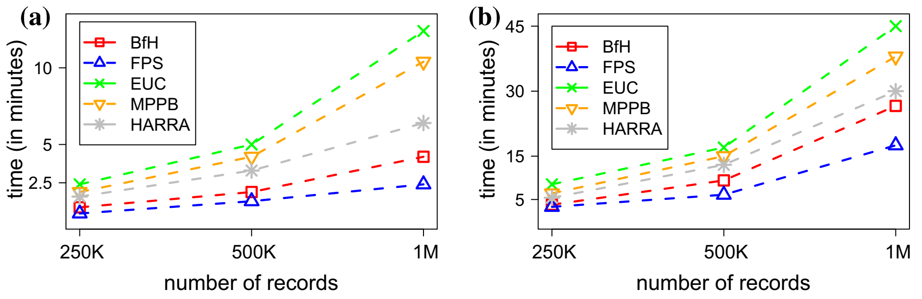


Fig. 10 Measuring the wall-clock times of all methods using the NCVR-based data set. **a** Applying Pt_1. **b** Applying Pt_2

6.3.8 Reduction ratio

The *RR* is high for all LSH-based methods (BfH, FPS, and HARRA), but this reduction in the comparison space is *efficient* only for FPS and BfH, which achieve also high *PC* rates, as can be seen in Fig. 9.

6.3.9 Wall-clock time

Figure 10a, b illustrates the wall-clock times for all methods by increasing the number of the records of the data sets at hand. In Pt_1, we observe that the wall-clock times of FPS, BfH, and HARRA increase almost linearly with the size of the data sets. We also notice that as the number of records increases, the gap between FPS/BfH/HARRA and the other methods also increases. By applying Pt_2, the larger number of blocking groups increases the wall-clock time of FPS, but its simple counting mechanism, which prevents most of the unnecessary distance computations, keeps it fairly lower than the other methods.

6.3.10 Increasing the number of parties

Finally, we performed a series of experiments by using three different NCVR-based data sets, which are A', B', and C' and materialize the scenario of three different data custodians. We perturb these data sets, which are of varying size, by applying the Pt_1 and Pt_2 schemes. Under this multi-custodian scenario, a pair might be formulated by Bloom filters which belong to any of these three data sets. When using $1M$ of records for each data set, it was impossible

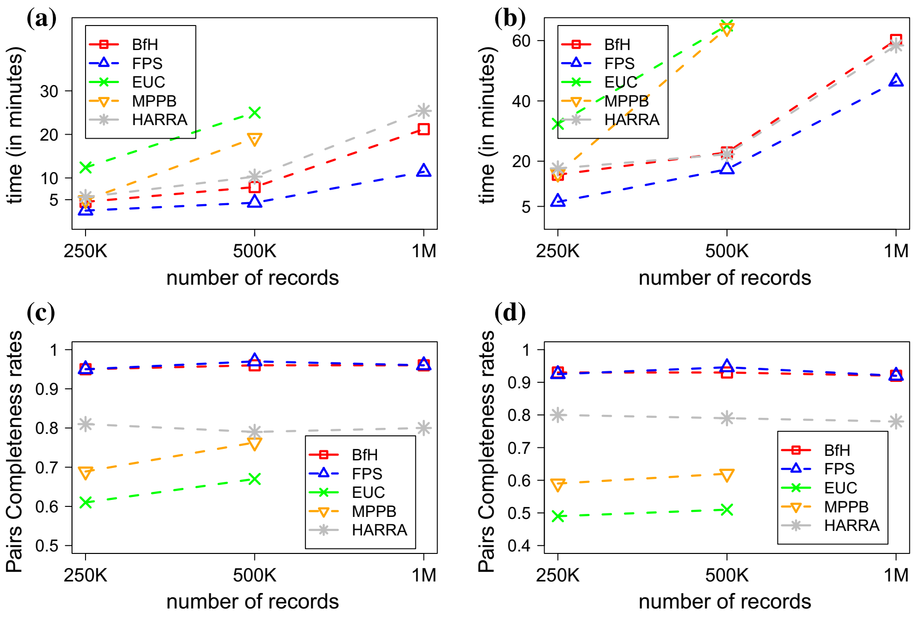


Fig. 11 Measuring the wall-clock times and the *PC* rates of all methods using three NCVR-based data sets. **a** Applying Pt_1. **b** Applying Pt_2. **c** Applying Pt_1. **d** Applying Pt_2

to obtain any measurements for the EUC and MPPB methods due to their excessive space requirements in main memory. FPS first stored the *Id*'s of the Bloom filters of A' and then counted the collisions for pairs formulated by using B', and C',[12] as discussed in Sect. 5.2. The scalable performance of FPS is clearly shown in Fig. 11a, b where we observe that the slope of its line is increasing, but is definitely smoother than the other methods. We also notice that the performance of BfH and HARRA is almost the same. The corresponding *PC* rates shown in Fig. 11d exhibit the superiority of FPS and BfH in identifying the similar pairs as also illustrated in Fig. 8a, b. A common and natural consequence for all methods is the increase in the space requirements, where the LSH-based blocking methods (FPS, BfH, HARRA, and EUC) need to store the *Id*'s of A' and B', while MPPB should store the *Id*'s of all the participating data sets.

7 Conclusions

Linking huge collections of anonymized records is an intriguing problem in the core of the domain of Privacy-Preserving Record Linkage. In this paper, we introduce a novel scheme, that is FPS, which accomplishes high levels of recall, by performing the least possible distance computations. FPS formally relies on the number of collisions that Bloom filter pairs achieve in the buckets of the blocking groups and provides theoretical performance guarantees based on the confidence parameter δ. We also quantify (a) the number of blocking that are required and (b) the exact number of collisions that a pair should exhibit in order to be considered for comparison.

[12] Counting the collisions for C' required storing the *Id*'s of both A' and B'.

For the future, we intend to investigate the applicability of FPS on multi-party settings in order to eliminate the possibility of collusion among the participating parties. Another fruitful research direction is the application of our scheme to the Map/Reduce framework [8]. Within this framework, we plan to (a) implement hash tables, which materialize the blocking groups, in a distributed manner, and (b) use a storage system, which facilitates efficient queries to massive data sets. Applying FPS on real-time settings, where accuracy and speed are the most important factors, is another research direction of great interest.

Appendix: Solving for $L_\mathcal{C}$ in (9)

We bound above the right-hand side of (9) in order to guarantee that the probability for a pair with $d_H = \vartheta$ to exhibit less collisions than $\mathcal{C}$ is bounded by δ as follows:

$$\exp\left\{-\frac{(L_\mathcal{C}\ p_\vartheta^K - (\mathcal{C}-1))^2}{2\ L_\mathcal{C}\ p_\vartheta^K}\right\} \leq \delta \Leftrightarrow -\frac{(L_\mathcal{C}\ p_\vartheta^K - (\mathcal{C}-1))^2}{2\ L_\mathcal{C}\ p_\vartheta^K} \leq \ln(\delta). \qquad (11)$$

We expand $(L_\mathcal{C}\ p_\vartheta^K - (\mathcal{C}-1))^2$ and derive the following quadratic equation:

$$(p_\vartheta^K\ L_\mathcal{C})^2 + (2\ p_\vartheta^K\ \ln(\delta) - 2\ p_\vartheta^K(\mathcal{C}-1))L_\mathcal{C} + (\mathcal{C}-1)^2 \geq 0, \qquad (12)$$

where we finally solve for $L_\mathcal{C}$ and keep only the equality, since we want to be as optimal as possible.

References

1. Aggarwal CC, Yu PS (2000) The igrid index: reversing the dimensionality curse for similarity indexing in high dimensional space. In: International conference on knowledge discovery and data mining, pp 119–129
2. Andoni A, Indyk P (2008) Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun ACM 51(1):117–122
3. Bonomi L, Xiong L, Chen R, Fung BCM (2012) Frequent grams based embedding for privacy preserving record linkage. In: International conference on information and knowledge management, pp 1597–1601
4. Broder AZ, Charikar M, Frieze A, Mitzenmacher M (1998) Minwise independent permutations. In: Symposium on theory of computing, pp 327–336
5. Christen P (2012a) A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans Knowl Data Eng 12(9):1537–1555
6. Christen P (2012b) Data matching—concepts and techniques for record linkage, entity resolution, and duplicate detection. Springer, Data-Centric Systems and Applications
7. Datar M, Immorlica N, Indyk P, Mirrokni VS (2004) Locality-sensitive hashing scheme based on p-stable distributions. In: Symposium on computational geometry, pp 253–262
8. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
9. Durham E (2012) A framework for accurate efficient private record linkage. PhD thesis, Vanderbilt University, USA
10. Durham E, Kantarcioglu M, Xue Y, Toth C, Kuzu M, Malin B (2014) Composite Bloom filters for secure record linkage. IEEE Trans Knowl Data Eng 26(12):2956–2968
11. Dwork C (2006) Differential privacy. In: Automata, languages and programming, international colloquium. Springer, Berlin Heidelberg, pp 1–12
12. Gan J, Feng J, Fang Q, Ng W (2012) Locality-sensitive hashing scheme based on dynamic collision counting. In: International conference on management of data, pp 541–552
13. Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: International conference on very large databases, pp 518–529

14. Goodman J, O'Rourke J, Indyk P (2004) Handbook of discrete and computational geometry. CRC, Boca Raton
15. Hall R, Fienberg SE (2010) Privacy-preserving record linkage. In: International conference on privacy in statistical databases, pp 269–283
16. Hernandez MA, Stolfo SJ (1998) Real world data is dirty: data cleansing and the merge/purge problem. Data Mining Knowl Discov 2(1):9–37
17. Inan A, Kantarcioglou M, Bertino E, Scannapieco M (2008) A hybrid approach to private record linkage. In: International conference on data engineering, pp 496–505
18. Inan A, Kantarcioglu M, Ghinita G, Bertino E (2010) Private record matching using differential privacy. In: International conference on extending database technology, pp 123–134
19. Karakasidis A, Verykios VS (2012) A sorted neighborhood approach to multidimensional privacy preserving blocking. In: International conference on data mining workshops, pp 937–944
20. Karapiperis D, Verykios VS (2013) A distributed framework for scaling up LSH-based computations in privacy preserving record linkage. In: Balkan conference in informatics, ACM, pp 102–109
21. Karapiperis D, Verykios VS (2014) A distributed near-optimal lsh-based framework for privacy-preserving record linkage. Comput Sci Inf Syst 11(2):745–763
22. Karapiperis D, Verykios VS (2015) An LSH-based blocking approach with a homomorphic matching technique for privacy-preserving record linkage. IEEE Trans Knowl Data Eng 27(4):909–921
23. Kaufman L, Rousseeuw P (1987) Clustering by means of medoids. In: Statistical Data Analysis Based on the L1Norm, Reports of the Faculty of Mathematics and Informatics. Delft University of Technology. Elsevier, Amsterdam, pp 405–406
24. Kim H, Lee D (2010) Fast iterative hashed record linkage for large-scale data collections. In: International conference on extending database technology, pp 525–536
25. Kuzu M, Kantarcioglou M, Durham E, Malin B (2011) A constraint satisfaction cryptanalysis of Bloom filters in private record linkage. In: International conference on privacy enhancing technologies, pp 226–245
26. Kuzu M, Kantarcioglu M, Inan A, Bertino E, Durham E, Malin B (2013) Efficient privacy-aware record integration. In: International conference on extending database technology, pp 167–178
27. Motwani R, Raghavan P (1995) Randomized algorithms. Cambridge University Press, Cambridge
28. Niedermeyer F, Steinmetzer S, Kroll Martin M, Schnell R (2014) Cryptanalysis of basic Bloom filters used for privacy preserving record linkage. J Priv Confid 6(2)
29. Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. In: Eurocrypt, pp 223–238
30. Pang C, Gu L, Hansen D, Maeder A (2009) Privacy-preserving fuzzy matching using a public reference table. Intell Patient Manag 189:71–89
31. Rajaraman A, Ullman JD (2010) Mining of massive datasets, chapter finding similar items. cambridge University Press, Cambridge
32. Scannapieco M, Figotin I, Bertino E, Elmagarmid AK (2007) Privacy preserving schema and data matching. In: International conference on management of data, pp 653–664
33. Schnell R, Bachteler T, Reiher J (2009) Privacy-preserving record linkage using Bloom filters. BMC Med Inform Decis Making 9(1)
34. Schnell R, Bachteler T, Reiher J (2011) A novel error-tolerant anonymous linking code. Tech. report WP-GRLC-2011-02, German Record Linkage Center
35. Vatsalan D, Christen P, Verykios V (2011) An efficient two-party protocol for approximate matching in private record linkage. In: Australasian data mining conference, pp 125–136
36. Vatsalan D, Christen P, Verykios V (2013a) Efficient two-party private blocking based on sorted nearest neighborhood clustering. In: International conference on information and knowledge management, pp 1949–1958
37. Vatsalan D, Christen P, Verykios VS (2013b) A taxonomy of privacy-preserving record linkage techniques. Inf Syst 38(6):946–969
38. Weber R, Schek H, Blott S (1998) A quantitative analysis and performance study for similarity search methods in high dimensional spaces. In: International conference on very large data bases, pp 194–205
39. Yakout M, Atallah MJ, Elmagarmid AK (2009) Efficient private record linkage. In: International conference on data engineering, pp 1283–1286

Dimitrios Karapiperis is a PhD student with the Hellenic Open University. He also holds an MSc degree from the University of York (UK) and a BSc degree from the Technological Institute of Thessaloniki (Greece). His research interests lie in the areas of privacy-preserving record linkage, similarity algorithms and data structures, and approximation schemes which rely on various randomization schemes. Specifically, he deals with running time optimizations by applying the Locality-Sensitive Hashing technique to voluminous data sets which have undergone an anonymization process. Other particular research interests include the development of efficient privacy-preserving summarization algorithms, scalable solutions using the Map/Reduce programming paradigm, and distributed architectures.

Vassilios S. Verykios received the Diploma degree in Computer Engineering from the University of Patras, Greece in 1992, and the MSc and the PhD degrees from Purdue University, USA in 1997 and 1999, respectively. From 1999 to 2002 he was with the Faculty of Information Systems in the College of Information Science and Technology at Drexel University, USA, as a tenure track Assistant Professor. From 2002 to 2005 he held various research and academic positions at Intracom SA, SingularLogic SA, CTI, University of Thessaly, Hellenic Open University, University of Patras and the Athens Information Technology institute in Greece. From 2005 to 2011 he was an Assistant Professor in the Department of Computer and Communication Engineering at the University of Thessaly in Volos, Greece. Since January of 2011, he is an Associate Professor in the School of Science and Technology and the Director of the Graduate Program on Information Systems, both at the Hellenic Open University (HOU). From May of 2014 he serves as the Director of the e-Comet Lab (http://eeyem.eap.gr/en) at HOU.