Datenbank Spektrum (2016) 16:227–236
DOI 10.1007/s13222-016-0222-9

FACHBEITRAG

Speeding up Privacy Preserving Record Linkage for Metric Space Similarity Measures

Ziad Sehili[1] · **Erhard Rahm**[1]

Received: 5 February 2016 / Accepted: 28 April 2016 / Published online: 1 June 2016

Abstract The analysis of person-related data in Big Data applications faces the tradeoff of finding useful results while preserving a high degree of privacy. This is especially challenging when person-related data from multiple sources need to be integrated and analyzed. Privacy-preserving record linkage (PPRL) addresses this problem by encoding sensitive attribute values such that the identification of persons is prevented but records can still be matched. In this paper we study how to improve the efficiency and scalability of PPRL by restricting the search space for matching encoded records. We focus on similarity measures for metric spaces and investigate the use of M-trees as well as pivot-based solutions. Our evaluation shows that the new schemes outperform previous filter approaches by an order of magnitude.

Keywords Metric Space · M-Tree · Triangle Inequality · Bloom Filter · Record Linkage

This work was funded by the German Federal Ministry of Education and Research within the project Competence Center for Scalable Data Services and Solutions (ScaDS) Dresden/Leipzig (BMBF 01IS14014B).

✉ Ziad Sehili
sehili@informatik.uni-leipzig.de

Erhard Rahm
rahm@informatik.uni-leipzig.de

[1] Institut für Informatik, Universität Leipzig, 04009 Leipzig, Germany

1 Introduction

Big Data applications in research, administration and business increasingly require the processing, analysis or publishing of large amounts of person-related data. These tasks have to ensure a high degree of privacy, i.e., the right of individuals to determine by themselves when, how and to what extent information about them is communicated to others [1]. For this purpose, a large number of techniques has been proposed in particular for privacy-preserving publishing, record linkage and data mining of sensitive data [20, 21, 9]. We focus on *Privacy Preserving Record Linkage (PPRL)*, i.e., the privacy-preserving identification of records in different datasets referring to the same person. Use cases for PPRL include the integration of patient-related data from different sources, e.g. about certain diseases and their treatment, or the analysis of financial transactions and tax records to identify potentially criminal acts.

To ensure privacy, the parties involved in a PPRL approach (i.e., the data owners and possibly a dedicated linkage unit) must not reveal sensitive information that would allow the identification of persons. On the other hand, it should still be possible to match and combine records referring to the same person, which speaks against a complete anonymization of records, e.g., based on approaches such as k-anonymity and its variations [9]. Instead, PPRL methods apply a secure one-way *pseudonymization* on identifying attributes, the so-called quasi-identifiers, to achieve both privacy and the ability to link records. The quasi-identifiers such as name, address or birth date are not unique and can contain errors but in combination they are suitable to identify persons and thus need to be protected. For this purpose, most recent PPRL approaches use a set of one-way hash functions to map the quasi-identifiers into bit vectors (or bloom filters) of fixed sizes [18]. This approach has

been shown to be secure against re-identification of persons while still allowing an approximate matching since persons with similar quasi-identifiers also have similar bit vectors. In this paper, we will also follow such an encoding of sensitive data. Furthermore, we assume that the matching using the encoded bit vectors is done on a trusted third party to avoid exchanging data between data owners.

Like traditional approaches for record linkage (RL), PPRL has an inherent scalability problem if one matches each encoded record with all other encoded records (quadratic complexity w.r.t. number of records). Hence it becomes necessary to adapt the proven RL performance techniques to PPRL such as the reduction of the search space by filter and blocking techniques or parallel matching [6]. Here, we focus on filter techniques that utilize characteristics of common similarity functions and a predefined similarity threshold to limit the number of records to be matched. While this approach is quite common for RL [10] it has received relatively little attention for PPRL. Two of the most promising schemes so far are adaptations of the so-called multibit tree and PPJoin approaches [19].

In this paper, we study PPRL filter techniques for metric space similarity measures that are quite common. In particular we make the following contributions:

- We propose different approaches to utilize the so-called triangle inequality of metric spaces to reduce the search space for PPRL, in particular the use of M-trees as well as pivot-based solutions.
- We comparatively evaluate the different approaches to identify the most promising PPRL approach for metric space similarity measures.
- We evaluate and compare the new approach with previously proposed PPRL approaches based on multibit trees [2] and P4Join [19]. The evaluation shows that the new approach performs one magnitude of order faster so that we propose a highly promising technique for scalable PPRL.

After a discussion of related work we outline the assumed encoding scheme and PPRL architecture in Sect. 3. In the main Sect. 4 we present different approaches for efficient PPRL in metric spaces by using M-tree and pivot-based solutions. In the evaluation Sect. 5 we analyze the runtime efficiency of the metric space approaches and compare them with previous approaches. Finally, we conclude.

2 Related Work

The problem of *record linkage*, i.e., the identification of records representing the same real-word entity, has been addressed by numerous approaches and several surveys [13, 6, 8]. The main challenges are achieving both high match quality to find all matching records as well as high efficiency and scalability to large datasets. For high match quality it is generally necessary to consider the similarity of several identifying attributes such as title and authors for publications or name, address and sex for persons. Efficiency and scalability can be improved by blocking and filter techniques to reduce the search space as well as by executing the approaches on multiple processors in parallel (e.g., using Hadoop [12]).

Filter strategies are especially utilized for so-called similarity joins [10] requiring that the similarity of two records exceeds a minimal threshold to be a likely match. The PPJoin approach [22] is an efficient filter technique for the Jaccard string similarity as it employs several filters, e.g. to prune records from matching if they have high length difference or an insufficient overlap of tokens. The characteristics of similarity functions for metric spaces, in particular the triangle inequality (see next section) has also been utilized in previous approaches to reduce the search space for similarity search and record linkage, e.g., [15, 23]. Furthermore, dedicated index structures such as M-tree [7] have been proposed to facilitate a fast similarity search in metric spaces.

Privacy-preserving record linkage (PPRL) performs record linkage on encoded attribute values such as bloom filters [18] to ensure a high degree of privacy. PPRL has found substantial interest and a large number of approaches has been classified in a recent survey [21]. These approaches involve two or more data owners and either use a dedicated and trusted linkage unit or apply a symmetric protocol where encoded records are matched at the data owners. As most previous approaches, we study the use of a trusted linkage unit as it is less complex than symmetric protocols and avoids the exchange of records between data owners. PPRL approaches should be immune against different kinds of attacks to ensure a high degree of privacy. The encoding of sensitive information should not allow the re-identification of persons or specific attributes by *cryptoanalysis* or *frequency attacks* [16], e.g., by analyzing the frequencies and distributions of set bits in bloom filters and comparing these to known frequencies of attribute values such as last names. Furthermore, *collusion attacks* need to be prevented where different parties, e.g. the data owners and a linkage unit, violate the PPRL protocol and exchange sensible information such as the encoding approach. Finally, there is the risk of *dictionary attacks* where a leaked encoding scheme is applied to a dictionary such as a publicly known person directory (e.g., phone book or voter registry) to find specific persons.

For efficiency and scalability PPRL can in principle apply similar blocking, filtering and parallelization techniques than regular linkage approaches but need to be adapted to work with encoded data and its similarity measures. With

respect to filtering the use of so-called multibit trees, originally proposed for fast querying large databases of chemical *fingerprints* represented by bit vectors [14], has been shown to perform well for PPRL with bloom filters and even better than several blocking approaches [2]. An adaptation of PPJoin for PPRL called P4Join (privacy-preserving PPJoin) has been proposed in [19] and was found to be similarly fast than the use of multibit trees; a parallel implementation on GPUs achieved substantial runtime improvements. We will consider these two methods (multibit tree and P4Join) in our comparative evaluation with the new metric space methods. One of the few previous PPRL studies for metric spaces is [17]. They did not apply bloom filters for encoding of records but mapped the attribute values of records into a metric space to mask the original values; they then applied Euclidean distance to identify similar records but without focusing on reducing the search space as we do.

3 PPRL setting

We briefly outline the assumed PPRL setting in this section, in particular the encoding with bloom filters and the three-party architecture with two data owners and a trusted linkage unit.

3.1 Encoding with Bloom Filter

For data encoding we follow the Bloom filter approach proposed by Schnell et al. [18]. In this approach the original values of all attributes to be used for matching are tokenized into a set of n-grams which are then mapped into a common bit vector (array) of fixed size l. Specifically, each n-gram (of each n-gram set) is hashed to *multiple* bits by applying k independent hash functions, each defining an index position of a bit which is set to 1. This can be achieved by a double hashing scheme combining two independent base hash functions f and g to determine the k hash values $h_1(x), ..., h_k(x)$ for each n-gram x [11]:

$$h_i(x) = (f(x) + i \cdot g(x)) \mathrm{mod} l. \quad (1)$$

As base hash functions the authors proposed the usage of two keyed hash message authentication codes (HMACs), namely, HMAC-SHA1 and HMAC-MD5 for f and g, respectively. The mapping of records and their n-grams to bit arrays is illustrated in Fig. 1 for the two names *tomas* and *tommas*.

The security aspect of Bloom filter was also discussed by the authors [18]. They analyzed the weaknesses of this encoding and presented methods to harden it against different kinds of attacks.

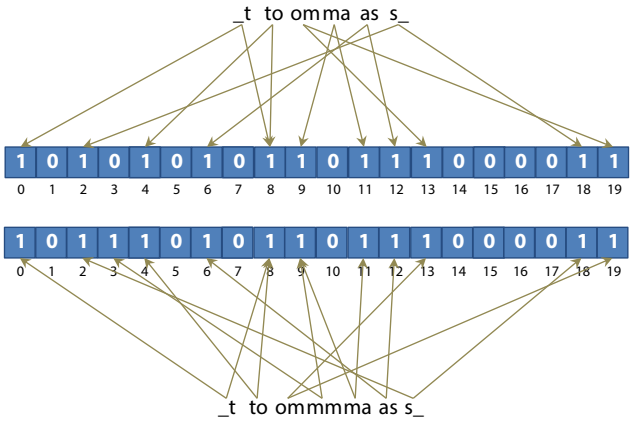

Fig. 1 Bit vector encoding of two names tomas and tommas, each tokenized to bigrams, using $k = 2$ hash functions and bit vectors of length $l = 20$

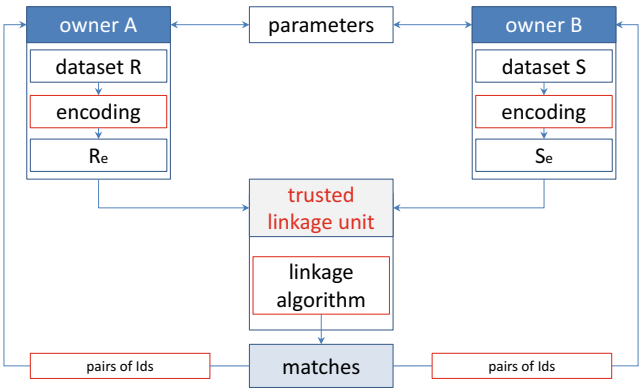

Fig. 2 Three-party protocol for PPRL

3.2 Use of Linkage Unit for PPRL

As in previous PPRL methods [21] we follow a so-called three-party protocol where the datasets R and S from two data owners A and B are matched in a privacy-preserving manner by a trusted third party or linkage unit (see Fig. 2). We also assume a so-called *Honest-but-Curious* cooperation scheme where each party tries to know as much as possible about the data of the other party while following the predefined protocol to perform record linkage. In particular there is no collusion between any data holder and the linkage unit. Thus, the data owners have to agree upfront on the PPRL parameters, in particular the attributes to be used for record linkage and their encoding. After exchanging this information the data owners encode their datasets (resulting into R_e and S_e) and send the masked data to the linkage unit for matching. The linkage unit only returns the *IDs* of the true matches to the data owners. Depending on the use case only the data for matching records can then be exchanged and combined for further analysis. The protocol thus avoids revealing information about non-matching

records, e.g. about patients that are not represented in both datasets. The linkage unit only sees encoded data that must not allow the identification of the underlying attribute values and persons.

4 PPRL for Metric Spaces

We start with explaining the triangle inequality of metric spaces and how it can be used to reduce the number of comparisons for record linkage. Next, we outline its use for matching bit vectors based on an approximation of the search radius. We then explain the use of M-tree and different pivot-based approaches for PPRL.

4.1 Metric Space

A metric space $\mathcal{M}(\mathcal{U}, d)$ consists of a set of data objects $\mathcal{U}$ and a metric d to compute the distance between these objects. Examples of such a metric are Minkowski distances (e.g., Euclidean distance), edit distance, Hamming distance and Jaccard coefficient [23]. These functions are commonly used for record linkage so that their optimization is of great interest. The main property satisfied by these distance functions and characteristic of metric spaces is the *triangle inequality*, which can be expressed as follow:

$$\forall x, y, z \in \mathcal{U} : d(x, z) \leq d(x, y) + d(y, z)$$

The triangle inequality is especially valuable as it can be used to reduce the search space for similarity search and record linkage. In both cases we have to find for a query object q those objects x with a distance $d(q, x)$ lower or equal a maximal distance threshold (or above a minimal similarity threshold) which can be seen as a radius $\text{rad}(q)$ around q (Fig. 3). The triangle equality allows one to avoid computing the distance between points y and q based on predetermined distances for y and q against reference points or pivots, such as point p in the figure. The triangle inequality implies that:

$$d(y, q) \geq d(p, q) - d(p, y).$$

If the difference between the precomputed distances $d(p, q)$ and $d(p, y)$ exceeds the maximal distance, i.e.,

$$d(p, q) - d(p, y) > \text{rad}(q),$$

we know that y and q cannot be similar enough so that we can avoid the distance computation $d(y, q)$. Analogously,

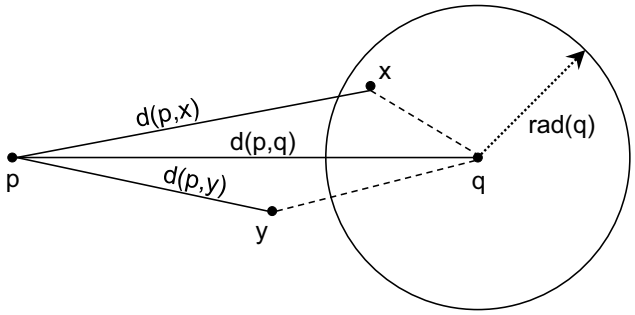

Fig. 3 Triangle inequality. Object y cannot lie within the search radius of query object q since the difference between $d(p, q)$ and $d(p, y)$ exceeds $\text{rad}(q)$

we only have to compute the distance $d(q, x)$ for objects x satisfying the triangle inequality expressed as:

$$d(p, q) - d(p, x) \leq \text{rad}(q) \tag{2}$$

The actual savings depend on the number and choice of pivots influencing the cost for predetermining the distances with the pivots as well as the number of objects assigned to a pivot (objects are always associated with their closest pivot). We will therefore study different alternatives to choose the pivots in Sect. 4.4.

4.2 Similarity and Search Radius for Bit Vectors

As shown in [18] the encoding scheme presented in Sect. 3.1 is similarity preserving for measures such as the Jaccard similarity. To determine likely matches we can thus use the bit vectors to find all pairs of records with a similarity above a threshold t. The Jaccard similarity between bit vectors x and q measures the degree of overlap between the set bit positions and is defined as follows:

$$\text{sim}_j(q, x) = \frac{|x \wedge q|}{|x \vee q|} = \frac{|x \wedge q|}{|x| + |q| - |x \wedge q|} \tag{3}$$

with $|x|$ denoting the number of set bits (or cardinality) in bit vector x. For the example of Fig. 1 the 11 set bits for the first name are also set in the second bit vector of cardinality 12. Hence, the Jaccard similarity is $11/12 = 0.92$.

For finding similar objects in metric space it is common to consider the distances between objects rather than their similarity. For a Jaccard similarity sim_j, this can be easily achieved by considering the distance $1 - \text{sim}_j$. We can also use alternate distance functions such as the *Hamming distance* which was shown to be equivalent to the Jaccard similarity [22]. The Hamming distance d_h measures the ab-

solute number of bit positions that differ in two bit vectors, i.e.,

$$d_h(q,x) = |x \vee q| - |x \wedge q| = |x| + |q| - 2\times|x \wedge q| \quad (4)$$

For the two bit vectors in Fig. 1 the hamming distance is 1. Since we want to use the Hamming distance in this paper we need to determine the maximal absolute distance (or query radius) for a query record q so that a certain relative similarity threshold t is met. For this we use the following equivalence relation between the Jaccard similarity and Hamming distance from [22]:

$$\text{sim}_j(q,x) \geq t \Leftrightarrow d_h(q,x) \leq (|q| + |x|)\times\left(\frac{1-t}{1+t}\right) \quad (5)$$

For the example in Fig. 1 and for a similarity threshold $t = 0.8$ we would obtain $d_h \leq (11 + 12)\times 0.2/1.8 = 2.56$. Hence only a hamming distance of 1 or 2 would meet the similarity threshold.

Equation 5 for the maximal hamming distance d_h refers to two input objects q and x. To determine a maximal distance (or radius) only with respect to a query object q we utilize the observation that only those points x with a cardinality $|x| \leq |q|/t$ can meet the similarity threshold t (length filter). By bringing this inequality into formula 5 we obtain the following approximation for the maximal hamming distance $d_h(q)$:

$$d_h(q) \leq \left(|q| + \frac{|q|}{t}\right)\times\left(\frac{1-t}{1+t}\right) = |q|\times\frac{1-t}{t}.$$

By using this approximation as the query radius $\text{rad}(q)$ we can utilize the triangular inequality in Eq. 2 to eliminate unsimilar objects. For the example in Fig. 1 and a similarity threshold $t = 0.8$ we obtain an approximate query radius (maximal hamming distance) of 2.75 and 3 for the first and second record, respectively.

4.3 PPRL with M-Tree

The M-tree is a dynamic index structure to organize the objects of a metric space based on the distances between objects and facilitates a fast similarity search [7]. For PPRL, we have to adapt the index structure to store bit vectors and to use a suitable distance measure such as the introduced Hamming distance. In our setting, the linkage unit first indexes the encoded objects of the first dataset R_e with the M-tree. For each encoded object q of the second dataset S_e the M-tree is then used to find the most similar R_e objects which are finally tested whether they match q.

The M-tree consists of two types of nodes: internal and leaf nodes both with a maximum number c of objects they store. An internal node p stores *routing objects*, that restrict a region of the metric space $\mathcal{M}$, and have the following format:

$$\text{routing}(p) = [p, ptr(T(p)), \text{rad}(p), d_h(p, \text{parent}(p))],$$

where p represents the bit vector of p, $ptr(T(p))$ is a pointer to the subtree covered by p, $\text{rad}(p)$ is the covering radius of p in the metric space, i.e., the maximal distance between p and the bit vectors in its subtree. Finally $d_h(p, \text{parent}(p))$ is the distance between p and its parent (routing) object.

Leaf nodes are used to store all bit vectors of objects or records r and have the following format:

$$\text{leaf}(r) = [r, id(r), d_h(r, \text{parent}(r))],$$

where r represents the bit vector of r, $id(r)$ its identifier and $d_h(r, \text{parent}(r))$ is the distance between r and its parent object (a routing object).

Algorithm 1: Insertion in M-tree

```
   Input: M-tree node N;
          x bit vector to insert;
1  if N is not a leaf then
2      let E be routing objects held by N;
3      find P ⊆ E s.t. ∀p_i ∈ P: d_h(p_i, x) ≤ rad(p_i);
4      if P ≠ ∅ then
5          choose p ∈ P s.t. ∀p' ∈ P: d_h(p, x) ≤ d_h(p', x);
6      else
7          choose p ∈ E s.t. ∀p' ∈ E:
             d_h(p, x) − rad(p) ≤ d_h(p', x) − rad(p');
8          set rad(p) = d_h(p, x);
9      insert(p, x);                    // recursive call
10 // N is a leaf node
11 else
12     let E be bit vectors held by N;
13     insert x in E;
14     if |E| = c + 1 then      // c: max node capacity
15         //split operation
16         promote(E, o_1, o_2);
17         partition(E, o_1, o_2);
```

Building Process: The design of the M-tree aims at storing similar objects together within few nodes so that similarity search can be restricted to a small subset of all objects. The building process (Algorithm 1) works in a bottom-up way and leads to a height balanced tree. To insert a new bit vector x the algorithm starts from the root and descends recursively through one of its subtrees until the leaf level. At each level the algorithm searches a routing object that can hold x without updating its radius. If there are several such routing objects the one closest to x is chosen (lines 3–5). If such a node does not exist, a routing object

that needs the minimal enlargement of its radius is chosen and its covering radius is updated to its distance to x (lines 7–8). The insertion of x in a leaf node leads to an overflow if the leaf contains already c bit vectors. In this case a split operation on the overflowed node is trigged by choosing two bit vectors o_1 and o_2 from the leaf (line 16) to be promoted as routing objects and then partitioning the $c+1$ bit vectors on o_1 and o_2 (line 17). This *promote* and *partition* operations may be propagated until the root, in which case a new root is designated and the tree grows up by one level.

In [7] several implementations of the functions *promote* and *partition* are discussed. In our implementation we keep the parent object of the overflowed node as the first routing object; as the second promoted object we choose the leaf object with the maximal distance to the parent object. Our partition function assigns each leaf object to the nearest promoted object.

Algorithm 2: Search in M-tree

Input : M-tree node N;
q query bit vector;
$maxd$ maximal hamming distance;
Output: M set of match pairs;

```
1  Let p be the parent of N;
2  M = ∅;
3  // Step 1: find potential routing objects
4  if N is not a leaf then
5      foreach r ∈ routing objects in N do
6          if |d_h(p,q) − d_h(p,r)| ≤ rad(r) + rad(q) then
7              compute d_h(r,q);
8              if d_h(r,q) ≤ rad(r) + rad(q) then
9                  Search(ptr(T(r)), q, maxd);
10 // Step 2: find similar records
11 else
12     foreach x ∈ bit vectors of leaf N do
13         if |d_h(p,q) − d_h(p,x)| ≤ rad(q) then
14             compute d_h(q,x);
15             if d_h(q,x) ≤ maxd then
16                 M = M + {(q,x)};
```

Search Process: The main goal of M-tree is to find the similar objects for a query object q within few nodes in order to reduce the amount of similarity computations. To achieve this goal M-tree utilizes the precomputed distances between (routing) objects and their parents and tries to filter out irrelevant nodes by applying the triangle inequality in two steps. Algorithm 2 shows how the similarity search for query object q and maximal distance max$d$ is performed. The search starts at the root and calls the procedure of Algorithm 2 for each node under the root which has a parent routing object in the root. In the first step the algorithm checks each routing object in the inner nodes whether its radius overlaps with the radius of the query object q by applying the triangle inequality (lines 6 and 8). The difference between the two forms of triangle inequality is that the first form approximates the radii overlap between the routing object r and query q by using p the parent of r. The second form is more accurate and starts by computing the distance between q and r (line 7) then testing whether their radii overlap or not. This process continues recursively until the leaves are reached. In the second step the bit vectors in the leaves are tested individually for possible similarity by applying the triangle inequality of Eq. 2. Only records that satisfy it are finally compared with q.

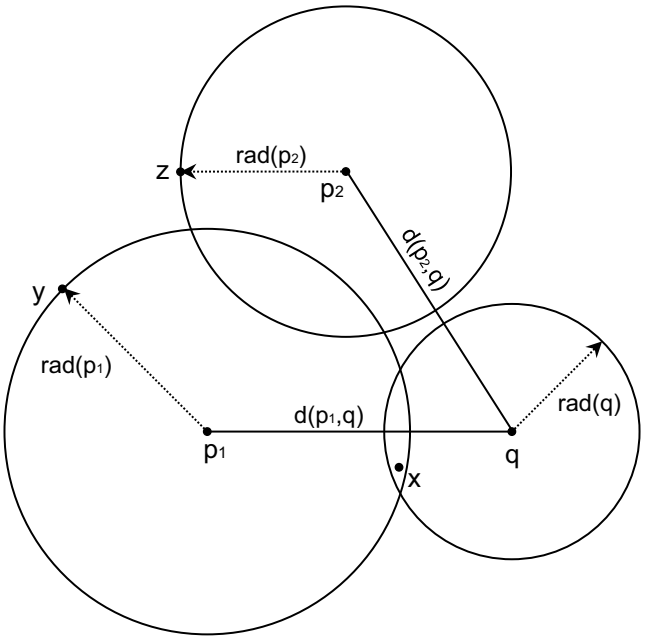

Fig. 4 The farthest object assigned to a pivot defines the radius of this pivot e.g. $\text{rad}(p_1) = d(p_1, y)$. Due to radii overlap the objects assigned to p_1 must be checked, but all the objects assigned to p_2 can be safely pruned

4.4 Pivot-based PPRL

The pivot-based PPRL approaches also work in two steps: preprocessing and similarity search. Preprocessing (Algorithm 3) entails selecting a certain number m of the data objects (of the first dataset R_e) as pivots or reference points. Furthermore each object x in R_e is assigned to its closest pivot p and the distance $d_h(p, x)$ and the maximal distance per pivot $\text{rad}(p)$ are precomputed. In Fig. 4, object y is farer away from pivot p_1 than all other objects assigned to p_1 so that $\text{rad}(p_1)$ is set to $d(p_1, y)$. Similarity search is performed for each object q in the second dataset S_e against the pivots and utilizes the triangle inequality to minimize the number of distance computations.

There are several possibilities to select the pivots incurring different preprocessing overhead and potentially different efficiency for similarity search. We consider three approaches that will be comparatively evaluated:

- *M-tree leaves:* We use a M-tree and select the routing parent object of each leaf as a pivot, i.e., we have as many pivots as leaves in the M-tree. The number of pivots is inversely proportional to the chosen node capacity of the M-tree. Since the M-tree places already every object in a leaf we do not require a new assignment of objects to pivots, i.e., we do not have to apply Algorithm 3 in this case.
- *Far-away pivots:* Pivots are iteratively determined from the set of all objects such that the object with the greatest distance to all previously determined pivots becomes the next pivot [3].
- *Sample-based:* This approach uses the far-away scheme but only on a sample of the objects to reduce the preprocessing overhead. As proposed in [4], we use a randomly determined sample of $3m$ objects, where m is the number of pivots.

Algorithm 3: Pivot-based preprocessing

Input : R_e set of bit vectors;
m number of desired pivots;

```
1 // Step 1: find pivots by selecting one of
     several possible methods
2 P = findPivots (R_e, m)
3 // Step 2: assign bit vectors to closest pivots
4 foreach x ∈ R_e \ P do
5     assign x to the nearest pivot p ∈ P;
6     store the distance d_h(p, x);
7     if d_h(p, x) > rad(p) then
8         rad(p) = d_h(p, x); // update pivot's radius
```

Algorithm 4: Far-away pivot selection

Input : R_e set of bit vectors;
m number of desired pivots;
Output: P set of pivots;

```
1 P = ∅;
2 pick a random record x ∈ R;
3 find p ∈ R_E : ∀p' ∈ R_e, d_h(x, p) > d_h(x, p');
4 P = P + {p};
5 while |P| < m do
6     find p ∈ R_e s.t. ∀p' ∈ R_e :
7     Σ_{i=0}^{|P|} d_h(p_i, p) > Σ_{i=0}^{|P|} d_h(p_i, p'), ∀p_i ∈ P;
8     P = P + {p};
9 return P
```

The rationale behind the far-away approaches is to select pivots that will probably lead to a low overlap of their regions so that the data space is well partitioned which can help restricting distance computations to few partitions. The approach is detailed in Algorithm 4. It starts from a randomly selected object (bit vector) and determines the farthest object from it as the first pivot (lines 3–4). The following pivots are chosen successively to be the bit vectors having the maximal distance from the already selected pivots (lines 6–8). The complexity of this pivot selection is $\mathcal{O}(nm)$ for n objects which is reduced to $\mathcal{O}(m^2)$ for the sample-based approach.

Algorithm 5: Search by means of pivots

Input : P set of pivots generated in algorithm 3;
$q \in S_e$ query bit vector;
$maxd$ maximal hamming distance;
Output: M set of match pairs;

```
1  M = ∅;
2  foreach p ∈ P do
3      // step 1: check radii overlap
4      if d_h(p, q) ≤ rad(p) + rad(q) then
5          foreach r assigned to p do
6              // step 2: check triangle inequality
7              if d_h(p, r) − d_h(p, q) ≤ rad(q) then
8                  // step 3: compare bit vectors
9                  if d_h(r, q) ≤ maxd then
10                     M = M + {(q, r)};
11 return M
```

Similarity search for a query object q includes three main steps as outlined in Algorithm 5. Initially all pivots p_i are determined that may contain similar records to q by checking whether the radii $\mathrm{rad}(p_i)$ and $\mathrm{rad}(q)$ overlap or not:

$$d(p_i, q) \leq \mathrm{rad}(p_i) + \mathrm{rad}(q) \tag{6}$$

If the radii of p_i and q do not overlap we know that *all* bit vectors assigned to p_i can be pruned so that similarity computations between them and q are saved.

In step 2, we check for the remaining pivots p_i for each of its assigned records r whether the triangle inequality holds w.r.t. q. If not, record r can safely be pruned from further consideration. Otherwise we have to compute the distance $d_h(q, r)$ and include r in the set of matches if it lies within the maximal distance from q (step 3).

For the example of Fig. 4 we can prune pivot p_2 and its objects in the first step. Pivot p_1 is further considered and its objects x and y both satisfy the triangle inequality in step 2. In the last step it is found that only x is close enough to q so that (q, x) is determined as a match.

5 Evaluation

We now evaluate the proposed methods for metric space PPRL and compare them with previously proposed filter techniques. After the description of the experimental setup we evaluate the use of M-trees and the three pivot-based methods. In Sect. 5.3, we present the comparison with P4Join [19] and Multibit trees [2].

5.1 Experimental setup

For our experiments we used the tool from [5] to generate five differently sized datasets of person records similarly as in previous studies, e.g., [19]. Each generated dataset of size n consists of two subsets R and S of sizes $4n/5$ and $n/5$ respectively as shown in Fig. 5. The records of S are obtained by introducing some common errors in 25 % of the R records. All records are encoded on three attributes (name, surname and birth date) by tokenizing the values into bigrams (for birth date unigrams). The tokens of each record are mapped to a bit vector of size 1000 using 20 hash functions. The similarity function used in Multibit tree and P4Join is Jaccard (Eq. 3) with a threshold $t = 0.8$. The analogous threshold is used for the hamming distance of the new metric space approaches.

Our evaluation focuses on the runtime efficiency and not match quality. We verified that all algorithms return the same set of matches due to the use of the same or equivalent similarity measures and thresholds.

All experiments are conducted on a desktop machine equipped with an Intel $i7-4770$ (3.4 GHz) CPU and 16 GB main memory and running Ubuntu 14.04.3.

	$\|D_1\|$=100,000	$\|D_2\|$=200,000	$\|D_3\|$=300,000	$\|D_4\|$=400,000	$\|D_5\|$=500,000
$\|S\| = \|D_i\|/5$	20,000	40,000	60,000	80,000	100,000
$\|R\| = \|D_i\| * 4/5$	80,000	160,000	240,000	320,000	400,000

Fig. 5 Sizes of the input datasets

Max node capacity	100	200 - 1500	1600 - 2000
Number of pivots	17,873	10,689 – 1,699	1,509 – 1,307
M-tree height	4	3	2

Fig. 6 Correspondence between the M-tree maximal node capacity c and the number of leaves representing pivots

5.2 M-Tree vs. pivot-based approaches

In the first experiment, we compare the efficiency of using the M-tree index structure and the three pivot-based PPRL approaches described in Sect. 4. We use the largest dataset $|D_5| = 500{,}000$ ($|S_e| = 100{,}000$ and $|R_e| = 400{,}000$) to run this experiment. For the sample-based pivot strategy we used a random sample of size $3m$ of bit vectors from R_e to choose m pivots from. The number of leaf nodes in the M-tree (and thus the number of pivots in the leaf-based approach) depends on the maximal node capacity. The table in Fig. 6 shows that a small capacity of 100 objects per node results in 17,873 leaves (pivots) and a tree of height 4. On the other hand larger nodes with 1,600 objects or more result in a tree of height 2 with only 1,509 leaves (pivots) or less.

Figure 7a shows the runtime results of the four alternatives for metric space PPRL for different numbers of pivots or leaf pages for the M-tree based approaches. To simplify the comparison we chose the number of pivots as they resulted for different node capacities for the M-tree (Fig. 6). To help explain the results, we also show the runtime portions for preprocessing (indexing, pivot selection) in Fig. 7b. We observe that for most choices of the num-

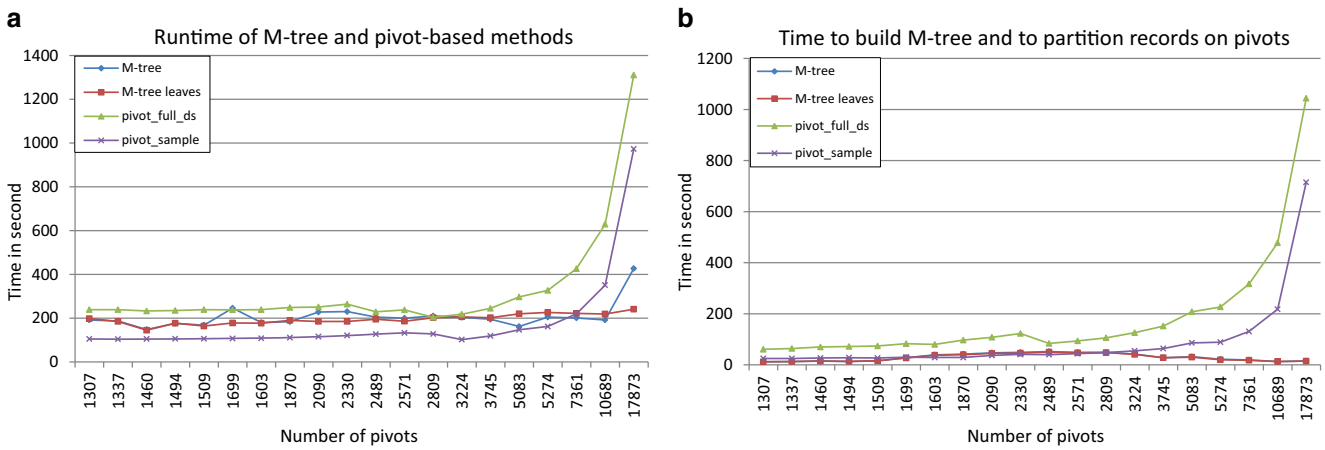

Fig. 7 Comparison of metric space PPRL approaches (M-tree and three pivot-based strategies). **a** Runtimes for M-tree and the pivot-based PPRL strategies, **b** Runtimes for preprocessing/indexing

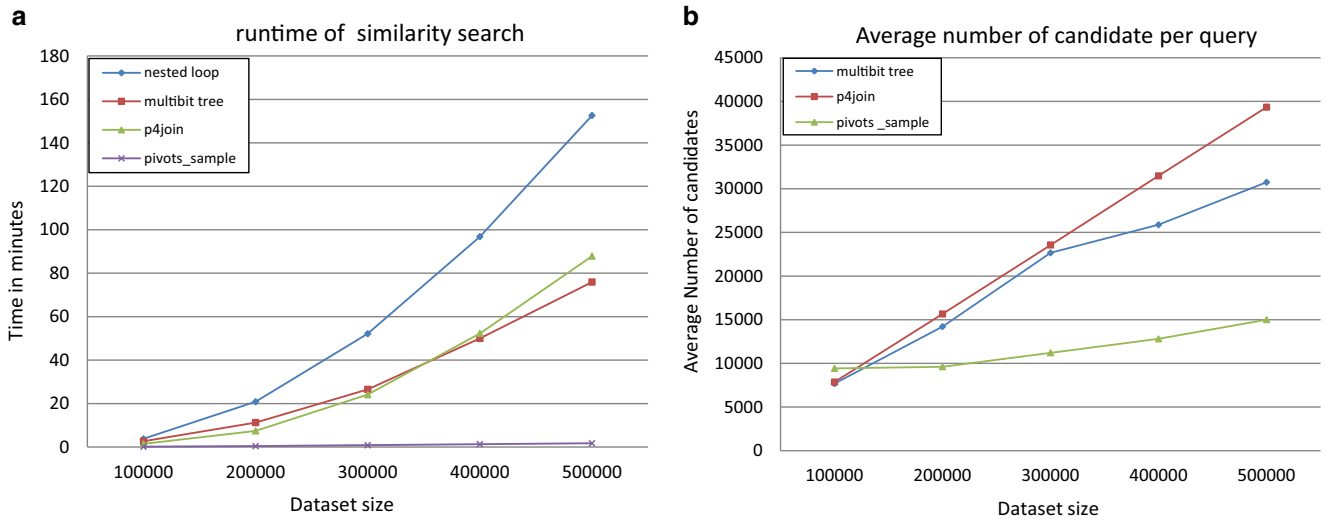

Fig. 8 Comparison of the pivot-based strategy *pivot_sample* with Multibit tree, P4Join and Nested Loop w.r.t. the runtime and the filtering effect. **a** Runtime in minutes of the different algorithms, **b** Average number of candidates per query bit vector

ber of pivots (up to about 4000 pivots) the runtimes for the different approaches are largely stable. In this range the sample-based pivot approach (*pivot_sample*) performs best with a runtime of mostly about 120 s (its optimum is 103 s) while the far-away pivot selection for the full dataset (*pivot_full_ds*) performs worst and needs almost twice as much time (230 s). Using M-tree and the leaf-based pivot selection perform very similar and achieve a medium performance. They remain largely stable even for a larger number of of pivots (>7,000) while the two other (far-away) pivot-based approaches suffer in this range from strong runtime increases.

This behavior is due to the high preprocessing overhead of the far-away approaches – both *pivot_full_ds* and *pivot_sample* – that grows at least quadratically with the number of pivots (Sect. 4.4). As can be seen in Fig. 7b, the preprocessing time of these two approaches grows sharply for more than 4,000 pivots and largely dominate the total runtime. Hence, the two approaches are dependent on a small to moderate number of pivots. By contrast the indexing time for M-tree is very small and virtually independent from the node capacity and thus the number of pivots. This however means that the time for similarity search – which is the difference between the total runtime and preprocessing time – is significantly better for the two far-away pivot approaches than using the M-tree. Better search times are the result of a higher filter effectiveness. Hence, we could prune out more pivots and their associated objects from the distance computation with the pivot-based approaches (especially for *pivot_sample*) than using the M-tree which thus seems to have suffered from substantial overlaps between the data regions of the leaf pivots and inner nodes. As expected, *pivot_sample* has a faster preprocessing than *pivot_full_ds*. However these differences are smaller than the differences in the total runtime. This implies that the search efficiency of *pivot_sample* is better than for *pivot_full_ds* despite that the pivots are selected from a much smaller objects set.

5.3 Comparison with previous filter techniques

We now study the relative performance of the metric space approaches with the two previously proposed Multibit tree and P4Join schemes. For metric space PPRL we focus on the best approach *pivot_sample* and report the runtimes for the best-performing number of pivots.

Figure 8a shows the runtime in minutes for each of the three algorithms as well as for a naive Nested loop approach without any filtering. We observe that the runtimes grow nearly quadratically with the dataset size not only for Nested Loop but also for the filter approaches. However, we see that the proposed pivot-based filtering dramatically outperforms both multibit trees and P4Join. While *pivot_sample* needs only about 2 minutes runtime for the largest dataset (D_5), Multibit tree and P4Join require about 76 and 88 minutes, respectively. Hence the new approach achieves a speedup of about 40 for this data size (the other metric space PPRL schemes also achieve an order of magnitude improvement as they were less than a factor 2 slower than *pivot_sample*). These unexpectedly high improvements were favored by the fact that both Multibit tree and P4Join achieved only modest improvements (less than a factor 2) compared to the naive Nested Loop indicating an insufficient filter effectiveness.

To illustrate the filter effectiveness we determined for each algorithm the number of candidate pairs to check for

a match and calculate the average number of candidates per query. As illustrated in Fig. 8b the number of candidates per query grows linearly with the data volume[1] and correlates with the runtime reported in Fig. 8a. The pivot-based approach has a much lower number of candidates per query than the other algorithms, especially for large datasets. For the largest dataset it generates only half as many candidates to check than for the Multibit tree. An additional advantage of the pivot-based method is its very low cost of pruning dissimilar bit vectors. This method uses one distance function and the triangle inequality to prune either one record or all records associated with a non-overlapping pivot.

To explore the scalability aspect a little further we doubled the data volume to 1 million bit vectors (80 % in the first and 20 % in the second dataset) for an additional test run. In this case we achieved a runtime of 5.2 minutes for *pivot_sample* compared to 216 minutes for Multibit tree and 345 minutes for P4Join. While the runtime for the pivot-based approach also increases more than linearly it remains 40 times faster than using Multibit tree and even 66 times faster compared to P4Join thus confirming the observations from Fig. 8a.

6 Conclusion

We studied the runtime optimization of Bloom filter-based privacy-preserving record linkage for metric space similarity measures. The main goal is to significantly reduce the search space by utilizing the triangle inequality for metric space measures such as Jaccard similarity or Hamming distance. We outlined the use of M-trees and three pivot-based approaches for PPRL. A thorough evaluation showed that the new schemes are highly effective and outperform previous filter techniques such as Multibit trees and P4Join by more than an order of magnitude. The best performing scheme is a pivot-based approach that selects the pivots from a relatively small random sample of all objects and that tries to minimize the overlap between the data regions of the pivots.

Despite the high filter effectiveness the runtimes for the new approaches still increase almost quadratically with the data size. As a result we have to further improve the runtime for PPRL to be able to deal with very large data volumes. We thus plan to combine the pivot-based filtering with blocking and parallel processing, either on GPUs or Hadoop-based platforms.

[1] Since the number of queries also grows linearly with the data volume, runtimes increase almost quadratically.

References

1. Agrawal R, Kiernan J, Srikant R, Xu Y (2002) Hippocratic databases. In: Proc. VLDB conf, pp 143–154
2. Bachteler T, Reiher J, Schnell R (2013) Similarity Filtering with Multibit Trees for Record Linkage. Tech. Rep. WP-GRLC-2013-01. German Record Linkage Center
3. Bozkaya T, Özsoyoglu ZM (1999) Indexing large metric spaces for similarity search queries. ACM Trans Database Syst 24(3):361–404
4. Brin S (1995) Near neighbor search in large metric spaces. In: Proc. VLDB conf, pp 574–584
5. Christen P (2005) Probabilistic Data Generation for Deduplication and Data Linkage. In: Proc. 6th Int. Conf. Intelligent Data Engineering and Automated Learning, pp 109–116
6. Christen P (2012) Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer
7. Ciaccia P, Patella M, Zezula P (1997) M-tree: An efficient access method for similarity search in metric spaces. In: Proc. VLDB conf, pp 426–435
8. Elmagarmid AK, Ipeirotis PG, Verykios VS (2007) Duplicate Record Detection: A Survey. IEEE Trans Knowl Data Eng 19(1):1–16
9. Fung B, Wang K, Chen R, Yu PS (2010) Privacy-preserving data publishing: A survey of recent developments. ACM Comput Surv 42(4):14
10. Jiang Y, Li G, Feng J, Li WS (2014) String similarity joins: An experimental evaluation. PVLDB 7(8):625–636
11. Kirsch A, Mitzenmacher M (2006) Less Hashing, Same Performance: Building a Better Bloom Filter. In: Proc. ESA Symp, pp 456–467
12. Kolb L, Thor A, Rahm E (2012) Dedoop: Efficient Deduplication with Hadoop. PVLDB 5(12):1878–1881
13. Köpcke H, Thor A, Rahm E (2010) Evaluation of entity resolution approaches on real-world match problems. PVLDB 3(1) 484--493
14. Kristensen TG, Nielsen J, Pedersen CNS (2010) A tree-based method for the rapid screening of chemical fingerprints. Algorithms Mol Biol 5:9
15. Ngomo ACN, Auer S (2011) Limes-a time-efficient approach for large-scale link discovery on the web of data. In: Proc. IJCAI
16. Niedermeyer F, Steinmetzer S, Kroll M, Schnell R (2014) Cryptanalysis of basic bloom filters used for privacy preserving record linkage. J Priv Confidentiality 6(2):59–79
17. Scannapieco M, Figotin I, Bertino E, Elmagarmid AK (2007) Privacy preserving schema and data matching. In: Proc.ACM SIGMOD conf, pp 653–664
18. Schnell R, Bachteler T, Reiher J (2011) A Novel Error-Tolerant Anonymous Linking Code. Tech. Rep. WP-GRLC-2011-02. German Record Linkage Center, Duisburg
19. Sehili Z, Kolb L, Borgs C, Schnell R, Rahm E (2015) Privacy preserving record linkage with PPJoin. In: Proc. BTW, pp 85–104
20. Vaidya J, Zhu Y, Clifton CW (2006) Privacy Preserving Data Mining. Advances in Information Security, vol. 19. Springer
21. Vatsalan D, Christen P, Verykios VS (2013) A taxonomy of privacy-preserving record linkage techniques. Inf Syst 38(6):946–969
22. Xiao C, Wang W, Lin X, Yu JX (2008) Efficient Similarity Joins for Near Duplicate Detection. In: Proc. 17th Int. Conf. on World Wide Web, pp 131–140
23. Zezula P, Amato G, Dohnal V, Batko M (2006) Similarity search: the metric space approach. Springer