German Record Linkage Center
WORKING PAPER SERIES
NO. WP-GRLC-2013-01 | MARCH 15, 2013

Similarity Filtering with Multibit Trees for Record Linkage

Tobias Bachteler | Jörg Reiher | Rainer Schnell

Similarity Filtering with Multibit Trees for Record Linkage

Tobias Bachteler, Jörg Reiher, and Rainer Schnell

German Record Linkage Center
University of Duisburg-Essen, D-47057 Duisburg, Germany
Rainer.Schnell@uni-due.de

15.3.2013

Abstract. Record linkage is the process of identifying pairs of records that refer to the same real-world object within or across data files. Basically, each record pair is compared with a similarity function and then classified in supposedly matching and non-matching pairs. However, if every possible record pair has to be compared, the resulting number of comparisons leads to infeasible running times for large data files. In such situations, blocking or indexing methods to reduce the comparison space are required.
In this paper we propose a new blocking technique (Q-gram Fingerprinting) that efficiently filters record pairs according to an approximation of a q-gram similarity function. The new method first transforms data records into bit vectors, the fingerprints, and then filters pairs of fingerprints by use of a Multibit Tree according to a user-defined similarity threshold.
We examined the effect of different parameter choices of Q-gram Fingerprinting, tested its scalability, and performed a comparison study including several alternative methods using simulated person data. The comparison study showed promising results for the proposed method.

Keywords: Blocking, Bloom-Filter, Indexing, Multibit Trees, Q-Grams, Record Linkage

1 Introduction

Record linkage is the process of identifying pairs of records that refer to the same real-world object within or across data files. Most often, record-linkage is done to improve data quality or enhance the available data of two or more files by merging them into one combined data file. The situation is simple when a unique linkage key is available. Otherwise, record linkage must rely on error prone proxy identifiers as names, addresses, or birth dates. Basically, each record pair is compared by a similarity function and then classified as supposedly matching or non-matching pair. The first problem of record linkage consists in the choice of a similarity threshold that minimizes the number of misclassifications. These problems are discussed in detail in several excellent recent textbooks and reviews of record linkage techniques, e. g. [1],[2],[3].

The second problem of record linkage is due to the large number of comparisons. For example, if each pair of records of two files has to be compared with a similarity function the resulting running times for large data files are intolerable. In such situations, methods to reduce the comparison space are needed. Such methods are usually referred to as blocking or indexing methods. All blocking or indexing methods identify a reduced set of candidate pairs for comparison and classification. A well performing blocking technique keeps all or nearly all true matching record pairs in the set of candidate pairs while removing most of the true non-matching record pairs. In the last years, several new approaches were proposed. However, as a recent comparison study

demonstrates, no blocking or indexing method performs clearly best or exhibits a nearly optimal blocking performance [4]. Therefore, the search for better blocking methods still continues. Recently, Christen [3] formulated a crucial property of a well functioning blocking method: A blocking method should produce a set of candidate pairs in such a way that only those record pairs are in the set whose similarity lies above a certain threshold and remove all other record pairs. We here propose a new blocking technique, which meets this criterion.

2 Related Work

There are a variety of blocking methods for reducing the number of record pairs that need to be compared in record linkage. An excellent and exhaustive survey of the literature including a comparison study can be found in the 2012 paper of Christen [4], or in his 2012 book on data matching [3]. We will concentrate on those approaches, which are considered as best performing in that study. In addition, we consider the recent approach by [5]. For details on other blocking or indexing methods as *Q-gram Indexing* [6], *Suffix Array Indexing* [7], *Robust Suffix Array Indexing* [8], [9], and *String Map Based Indexing* [10] we refer to [4].

Most common in record linkage is the traditional *Standard Blocking* (SB) (see [11], [12], and [2]). SB partitions both input files into mutually exclusive blocks according to the values of a blocking key. The blocking key is formed by the values of one or more linkage attributes. Only pairs of records from the same block in both files are compared. Since some matching record pairs might not have the same blocking key value, usually several blocking keys are successively applied. To increase the likelihood of identical blocking key values, often phonetic encodings or substrings of identifiers are used as key.

To overcome the need for exact matching blocking key values, Hernandez and Stolfo [13] introduced the *Sorted Neighbourhood Method* (SNM). Here, both input files are pooled and sorted according to a blocking key. A fixed sized window is then slided over the records. Two records from different input files form a candidate pair if they are in the window at the same time. Again, usually several blocking keys are used subsequently. Drawing on the basic idea of the sliding window, several more sophisticated variants of the SNM were subsequently proposed. Adaptive Sorted Neighbourhood approaches are intended to overcome the drawbacks following from the fixed window size of the basic SNM. Yan et al. [14] presented two SNM-variants where the window size is adaptively changed based on the distance between the sorted blocking key values. The first variant, called *Incrementally-Adaptive SNM*, enlarges the window as long as the distance between the first and the last key values remains below a certain threshold. If the distance is above the threshold, the window size is reduced. To get an optimal starting size of the next window, this size is determined on the basis of the size of the previous window and the actual distance between its first and last blocking key value. In the second variant, *Accumulatively-Adaptive SNM*, there is no single window but a series of windows that overlap by one record. If the distance between the first and last key value in the current window is below the threshold, the window is moved forward by one position and enlarged. This is repeated as long as the threshold is not exceeded. All record pairs found within this series of windows form candidate pairs. Another variant of the SNM is presented in [5]. The underlying logic of the *Duplicate Count Strategy* is to vary the window size according to the number of matches found within the current window. At the beginning, matches are searched within a window of size w. The next window size is determined depending on the number of matches found. The window size is increased for regions with many matches. In the basic *Duplicate Count Strategy* (DCS), w is increased by one as long as the number of matches per record pair compared exceeds the threshold $\frac{1}{w-1}$. In the more sophisticated variant, named *DCS++*, for each detected match the subsequent $w-1$ records are added to the current window instead of increasing the window.

Canopy Clustering (CC) [15] forms candidate pairs from those records placed in the same canopy. All records from both input files are pooled. The first canopy is created by choosing a record at random from this pool. This randomly chosen record constitutes the central point of the first canopy. All records that are within a certain loose distance l from the central point are added to the canopy. Then, the central point and any records in the canopy within a certain tight distance t from the former are removed from the record pool. Further canopies are built in the very same way as the first one until there are no more remaining records. The result is a set of potentially overlapping canopies. Pairs that can be formed from the records of the same canopy constitute the set of candidate pairs.

3 *Q*-gram Fingerprint Filtering

3.1 Building Blocks of the Method

Bloom Filters A Bloom filter is a data structure proposed by Bloom [16] for checking set membership efficiently [17]. It consists of a bit vector of length m with all bits initially set to 0. Furthermore, k independent hash functions $h_1, \ldots, h_k$ are defined, each mapping on the domain between 0 and $m-1$. To store the set $S = \{x_1, x_2, \ldots, x_n\}$ in the Bloom filter, each element $x_i \in S$ is hash coded using the k hash functions and all bits having indices $h_j(x_i)$ for $1 \leq j \leq k$ are set to 1. If a bit was set to 1 before, no change is made.

Q-grams A *q-gram* is a substring of length q within a string. For example, the string SMITH contains the 2-grams SM, MI, IT, and TH. The set $\{\,\mathrm{SM}, \mathrm{MI}, \mathrm{IT}, \mathrm{TH}\,\}$ is called the q-gram set of the string SMITH [18]. Customarily, 2-grams are called bigrams and 3-grams are called trigrams. The *q-gram similarity* [19] of two strings a and b is computed as

$$S_{q-gram} = \frac{2\,c}{(|a| + |b|)}, \tag{1}$$

where c is the number of q-grams occurring in both strings and $|a|$ denotes the cardinality of the q-gram set of string a.

Often, the strings are padded (i.e. prefixed and suffixed) by q-1 special characters before splitting them up into q-grams [20] to emphasize the first and last characters of the strings. For example, if the string SMITH is padded by blanks the resulting bigram set is {_S, SM, MI, IT, TH, H_ }.

With *positional q-grams* [21] the locations of the q-grams within the string are recorded as additional information. For example, when using the first character of each bigram as locational label, the positional bigram set is $\{\,\mathrm{\_S}_0, \mathrm{SM}_1, \mathrm{MI}_2, \mathrm{IT}_3, \mathrm{TH}_4, \mathrm{H\_}_5\,\}$. The positional q-grams X_i and Y_j are considered equal if and only if $X = Y$ and $i = j$.

Q-gram Fingerprints of Records A q-gram fingerprint $\boldsymbol{A}$ is a Bloom filter in which q-gram sets of all identifiers from record A are consecutively hashed. Again, all bits of the Bloom filter are initially set to 0. The first identifier, say first name, is split into q-grams and is stored using k_1 hash functions. Then the second identifier, say last name, is split into q-grams and stored using k_2 other hash functions. This is repeated for all identifiers in the record. The resulting (filled) Bloom filter is the q-gram fingerprint of the record. It is important to note that different hash functions are used for each identifier, that different q-gram types for the identifiers may be used, and finally that different numbers of hash functions of each identifier may be used. Figure 1 illustrates the transformation of a record into a Q-gram Fingerprint.

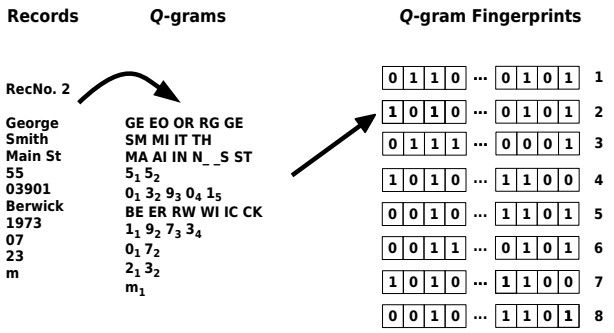

Fig. 1. Example of the transformation of a record into a Q-gram fingerprint. All identifiers of record No. 2 are split into bigrams or positional unigrams. The q-grams are then hashed into fingerprint No. 2. Fingerprints from other records are indicated to clarify that each record is represented by one fingerprint.

To store identifiers in fingerprints, we apply the double hashing scheme proposed by [22]. They show that only two independent hash functions are necessary to implement a Bloom filter with k hash functions without any increase in the asymptotic false positive probability. Therefore, k hash values are computed with the function

$$g_i(x) = (h_1(x) + ih_2(x)) \ \ mod \ m \tag{2}$$

where i ranges from 0 to $k-1$ and m is the length of the fingerprint. With our blocking method, we use two keyed hash message authentication codes, namely, HMAC-SHA1 (h_1) and HMAC-MD5 (h_2) [23] in the double hashing scheme. The choice of these HMACs is based only on their universal availability for nearly every programming language. Similarities between two q-gram fingerprints $\boldsymbol{A}$ and $\boldsymbol{B}$ can be determined by calculating the Jaccard similarity [19]

$$S_J(\boldsymbol{A}, \boldsymbol{B}) = \frac{\boldsymbol{A} \cap \boldsymbol{B}}{\boldsymbol{A} \cup \boldsymbol{B}}, \tag{3}$$

where $\boldsymbol{A} \cap \boldsymbol{B}$ is the number of bit positions set to 1 in both fingerprints (logical AND) and $\boldsymbol{A} \cup \boldsymbol{B}$ is the number of bit positions set to 1 in A or B (logical OR).

We approximate the q-gram similarity between two records by the Jaccard coefficient between their q-gram fingerprints. Hence the similarity filtering of records can be based on the Jaccard coefficients between their corresponding q-gram fingerprints.

Figure 2 illustrates the relationship between the two similarity measures. We took two lists of 1000 records comprising of first names, surnames, zip-codes and birth years.

Next we generated q-gram fingerprints using padded bigrams for the names and positional unigrams for zip-code and birth year, hashing each q-gram three times into a bit vector of length 1000. Then, we calculated the Jaccard coefficient for each possible pairing of fingerprints and their corresponding q-gram similarities from the original records. The plot shows box plots of the conditional distributions of Jaccard coefficients for q-gram similarities rounded to $0.0, 0.05, 0.1, \ldots, 0.95, 1.0$. The necessarily nonlinear, but monotone relationship between the two measures results in a Pearson correlation of 0.87 in this example.

Multibit Trees Kristensen et al. [24] introduced Multibit Trees[1] to search huge databases of molecular fingerprints efficiently. A molecular fingerprint summarizes structural information about

[1] See [25] for a mathematical analysis of Multibit Trees.

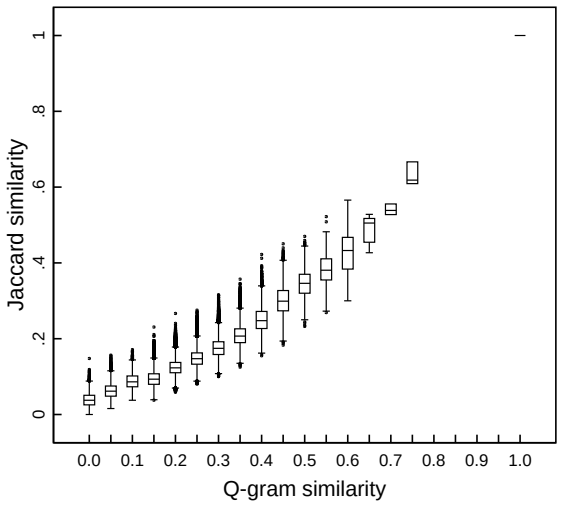

Fig. 2. Relationship between Jaccard coefficients of 1,000,000 pairs of q-gram fingerprints and the q-gram similarities of pairs of records represented by them.

molecules in a bit vector of length l. Molecular fingerprints are used to search for structural similar molecules.

Suppose one wants to search a query fingerprint $\boldsymbol{A}$ in a data base of molecular fingerprints $\boldsymbol{B}_i$. That is, one wants to find all fingerprints in the data base with a similarity to $\boldsymbol{A}$ above a certain threshold t. Multibit Trees find all fingerprints $\boldsymbol{B}_i$ where $S_J(\boldsymbol{A}, \boldsymbol{B}_i) \geq t$, or, equivalently, filters out all fingerprints below or equal to t.

Multibit Trees works with three steps: In a first step the fingerprint database is partitioned in groups of fingerprints. In the second step, within each group an actual Multibit Tree is built. In the third step, query fingerprints are searched in the trees.

In the *partition step*, all fingerprints $\boldsymbol{B}_i$ are grouped in bins of equal sized fingerprints. The size of a fingerprint, denoted by $|\boldsymbol{B}|$, is simply the number of bits set to 1 in $\boldsymbol{B}$. This allows to ignore all bins satisfying

$$|\boldsymbol{B}| \leq t\,|\boldsymbol{A}| \quad \text{or} \quad t|\boldsymbol{B}| \geq |\boldsymbol{A}| \tag{4}$$

in the searching step, because $\frac{min(|\boldsymbol{B}|,|\boldsymbol{A}|)}{max(|\boldsymbol{B}|,|\boldsymbol{A}|)}$ constitutes an upper bound on $S_J(\boldsymbol{A}, \boldsymbol{B})$.

In the *tree building step*, the fingerprints of equal size are stored in a binary tree structure: There is one tree for each bin. The storing algorithm starts with all fingerprints of the bin assigned to the root node of the tree. Then the algorithm recursively assigns at each parent node the fingerprints $\boldsymbol{B}_i$ having a 0 at a fixed bit position to the left subtree and all fingerprints having a 1 at that bit position to the right subtree. The determining bit position is chosen at each parent node such that the tree is kept as balanced as possible. Additionally, two lists of so called *match bits* are stored at each node. List O contains all bit positions with constant value 0 and list I all bit positions with constant value 1 in all remaining fingerprints below that node. The recursion stops if the number of fingerprints at a node falls below a previously defined threshold s.

For the query fingerprint $\boldsymbol{A}$, the *search step* is executed in three phases. In the *first searching phase*, all bins satisfying equation 4 are eliminated. In the *second searching phase*, a depth-first search is applied in each remaining tree. At each node of the tree currently searched, the recorded lists O and I allow the computation of an upper bound of the Jaccard Similarity for all fingerprints below the currently visited node. During the search, the algorithm determines

the quantities M_1 and M_0 for the query fingerprint $\boldsymbol{A}$ at each node. M_1 is the number of bit positions in $\boldsymbol{A}$ set to 1 and being member of list I. M_0 is the number of bit positions in $\boldsymbol{A}$ set to 1 and being member of list O. The bound on the Jaccard Similarity is then given by

$$S_J(\boldsymbol{A}, \boldsymbol{B}_i) \leq \frac{min(|\boldsymbol{A}| - M_0, |\boldsymbol{B}_i| - M_1)}{|\boldsymbol{A}| + |\boldsymbol{B}_i| - min(|\boldsymbol{A}| - M_0, |\boldsymbol{B}| - M_1)} \tag{5}$$

If the bounding quantity falls below the similarity threshold t, all nodes below can be eliminated from the search. In the *third searching phase*, the Jaccard Similarity between the query fingerprint $\boldsymbol{A}$ and all $\boldsymbol{B}_i$ stored at any external node not eliminated before is computed. If $S_J(\boldsymbol{A}, \boldsymbol{B}_i) \geq t$, the fingerprint is in the result set.

The partition step is intended to speed up the searching step beyond what can be achieved by the Multibit Tree structure alone. If the partition step is omitted, there is a single Multibit Tree built from all fingerprints in the database and the first searching phase can be omitted.

3.2 *Q*-gram Fingerprint Filtering with Multibit Trees

Using the building blocks from the previous section we can now describe the new blocking procedure. Let there be two data files to be linked. Blocking with Q-gram fingerprinting using Multibit trees can be done in three steps:

1. The records of both input files are transformed into Q-gram fingerprints.
2. The fingerprints of the larger file are stored in a Multibit tree.
3. Each fingerprint of the smaller file is queried against the Multibit tree.

Only records corresponding to fingerprints in the result set form candidate pairs with the record corresponding to the query fingerprint.

Successful application of all blocking methods require the choice of several parameters. For the new blocking method, six parameters have to be set. For the generation of q-gram fingerprints, one must decide on (a) the length m of the fingerprints, (b) the type of q-grams used for each identifier, and (c) the number of hash functions k_i for each identifier. For the querying phase, one must decide on (d) a similarity threshold t. For the Multibit tree setup, the recursion stop criterion s must be selected (e) and a decision on the use of the partition step before building the Multibit tree (f) must be made. Details of all parameter choices are discussed in section 4.4.

4 Experimental Evaluation

To evaluate the use of Q-gram fingerprints for blocking, we conducted three sets of experiments. The first set was designed to explore parameter settings. The second set of experiments examined the running times of Q-gram fingerprint filtering with increasing problem size. The third set of experiments compared the Q-gram fingerprint filtering to the best performing alternative blocking approaches in a recent study [4]. Additionally, we included a newly proposed and promising variant of the sorted neighbourhood approach [5] in the comparison study.

4.1 Test Data

All experiments were conducted using pairs of simulated test data files. The test data file was built from a sample of records from a national phone book. The records contained First Name, Surname, Street Name, House Number, Zip Code and Name of of the city. A Date of Birth was simulated according to the population distribution in 2006. Next, we used a precompiled list

recording first names with Sex. If a first name did not appear in the list, we assigned Sex with $p = .5$. The resulting list of simulated person records is the A-File in a test database.

For generating the B-File we used TDGen [26], an error simulation tool following the design principles of "generate2.py" [27].

TDGen allows to generate a second, garbled version of an existing test data file by introducing a variety of error types, including typos, phonetic errors, and OCR-errors. Error probabilities for the identifiers were estimated from a large national survey ($p = .15$ for First Name, $p = .2$ for Surname, $p = .2$ for Street Name, $p = .2$ for House Number, $p = .1$ for Zip Code, $p = .15$ for Name of the City, $p = .5$ for Date of Birth, and $p = .01$ for Sex). The error rates were applied to each field independently. After the introduction of errors, we split up Date of Birth into Birth Day, Month and Year.

For the experiments on the parameter selection for Q-gram fingerprint filtering (section 4.4) we used *Test Database I* with 50,000 records 40% of which contain errors in the B-file.
For the scalability experiment (section 4.4) we used *Test Databases II–XI* with 200,000, 400,000, ..., 2,000,000 records. In the comparison study (section 4.4), we used *Test Database XII* with 100,000 records and 40% erroneous rows and *Test Database XIII* with 100,000 records and 20% erroneous rows.

4.2 Hardware and Software

All experiments were conducted on a PC with a 2.8 GHz quadcore CPU and 16 GB of main memory, running 64-bit Ubuntu 10.04.

All procedures were implemented in Java as extensions to the KNIME platform [28]. For the Multibit tree algorithm we used the source code thankfully made available by Kristensen, Nielsen, and Petersen under `http://users-birc.au.dk/tgk/TanimotoQuery/`. For the implementation of the later discussed blocking technique DCS++ we used the Java code made available within the Duplicate Detection (DuDe) toolkit.[2]

4.3 Evaluation Metrics

The objective of a blocking method is the reduction of the comparison space while retaining as many true matching pairs as possible. We follow [5] to use *Recall* as the percentage of true matching pairs retained in the comparison space in relation to the required number of comparisons as evaluation measure of a blocking method. As second measure we use *Pairs Quality* defined as the ratio of true matching pairs C_M to the number of candidate pairs C_{M+NM} [4]:

$$PQ = \frac{C_M}{C_{M+NM}}. \tag{6}$$

Although the main factor determining running time is the number of comparisons, a blocking method by itself should not take too much time for its setup. We therefore measured the running time needed to to built the set of candidate pairs as third criterion (*Building Time*). Additionally, the average time required for searching the tree per query fingerprint (*Average Query Time*) was determined by querying 1000 randomly selected fingerprints.

[2] `www.hpi.uni-potsdam.de/naumann/projekte/dude_duplicate_detection.html`

4.4 Experiments

Selection of Parameters In section 3.2 we listed five parameters (a–e) for the setup of Q-gram Fingerprinting. We will discuss now each in turn. The length m of the fingerprints (parameter a) can be justified by the fact, that for optimal performance of Multibit trees, the number of 0s and 1s in the fingerprints should be similar [24]. Since for each q-gram and each hash function one bit position is set to one, we suggest to determine m as follows:

$$m = 2 \sum \bar{q}_i \, k_i, \tag{7}$$

where $\bar{q}_i$ is the average number of q-grams of and k_i the number of hash functions used for identifier i.

Regarding parameter (b), the type of q-grams used for each identifier, we see positional 1-grams for numerical identifiers as Zip Code, House Number, and Date of Birth as optimal choice. For names (including Street and Place Name), we examined two alternatives: (b 1) bigrams for names + positional 1-grams for numerical identifiers; (b 2) trigrams for names + positional 1-grams for numerical identifiers.

The choice of parameter (c), the number of hash functions k_i for each identifier, is related to the choice of fingerprint length, since the performance of the Multibit tree depends on the ratio of the number of 0s and 1s in the fingerprints. Moreover, recall and pair quality will depend on the quality of linkage identifiers. Therefore, it is obvious that identifiers with low error rates and high discriminating power [12], [29] should be weighted higher by hashing them with more hash functions [30]. We therefore tested three simple rules for determining k_i:

1. **Rule (c 1)**: Use a constant $k_i = 1$ for all identifiers.
2. **Rule (c 2)**: Use $k_i = 2$ for identifiers with few errors and high discriminating power and $k_i = 1$ for all others.
3. **Rule (c 3)**: Use 3 hash functions for identifiers with few errors and high discriminating power, 2 hash functions for identifiers with only high discriminating power and 1 hash function for identifiers with few errors only.

In many real world record linkage applications, Zip Code, Birth Year, Birth Month, Birth Day, Sex and House Number can be regarded as identifiers with low error rates. First Name, Surname, Street Name, and Place Name are in most applications found to be more error prone. The discriminating power of an identifier can be approximated by its entropy

$$H = -\sum_{i=1}^{m} p(x_i) \log_2 p(x_i). \tag{8}$$

Table 1 gives entropies calculated from test database I. With the exception of Sex, all identifiers show sufficient entropies and can therefore be regarded as having high discriminating power.

Table 2 lists the resulting number of hash functions k_i for each identifier given by the rules (c 1), (c 2), and (c 3).

Table 1. Entropies of the identifiers used in the experiments (Database I).

Identifier	Entropy
Sex	0.97
Birth Month	3.63
Birth Day	5.00
Birth Year	6.33
House Number	6.84
First Name	9.34
Place Name	11.19
Zip Code	12.46
Street Name	13.69
Surname	13.77

Table 2. Number of hash functions k_i for each identifier given by rules (c 1), (c 2), and (c 3)

Identifier	(c 1)	(c 2)	(c 3)
Sex	1	1	1
Birth Month	1	2	3
Birth Day	1	2	3
Birth Year	1	2	3
House Number	1	2	3
First Name	1	1	2
Place Name	1	1	2
Zip Code	1	2	3
Street Name	1	1	2
Surname	1	1	2

Figure 3 shows the number of non-matching comparisons and recall of the three different rules for k_i (Test Database I, bigrams). For Recall levels up to 0.995 the three rules yield very similar results. However, as the Recall level comes closer to 1.0, rule (c 3) results in fewer non matching comparisons. Figure 4 shows the same result for trigrams. Accordingly, we used rule (c 3) in all of the following experiments.

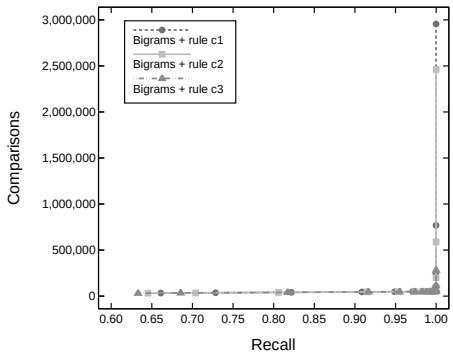

Fig. 3. Number of non-matching comparisons and recall of rules (c 1), (c 2), and (c 3) using bigrams

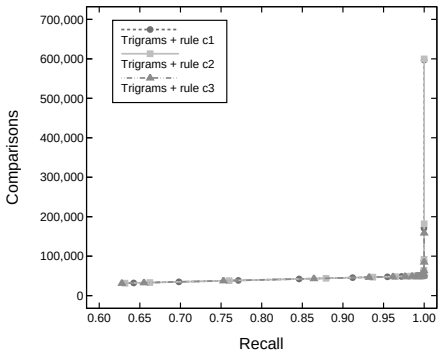

Fig. 4. Number of non-matching comparisons and recall of rules (c 1), (c 2), and (c 3) using trigrams

A direct comparison between bigrams and trigrams (both with fixed rule c 3), is shown in Figure 5. Trigrams yield substantially fewer non-matching comparisons at higher recall levels. It appears that trigrams should be used for identifiers like names with Q-gram Fingerprinting. Accordingly, all following tests are based on trigrams.

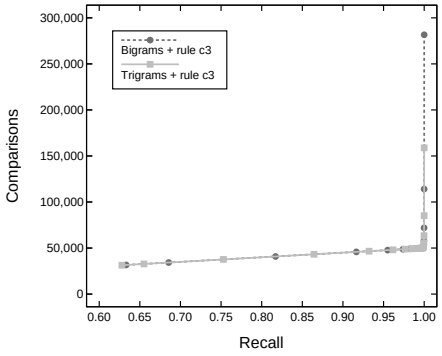

Fig. 5. Number of non-matching comparisons and recall for bigrams and trigrams

The choice of the similarity threshold t for the Multibit tree is probably the most important decision. Obviously, setting t to a low value will result in a high recall, but the number of remaining comparisons will be high as well. To illustrate this trade-off, figure 6 shows recall and pairs quality by similarity threshold t. Depending on the application, a user will have different loss-functions for weighting matching and non-matching pairs. In the rare case of equal weights, the obvious choice for t is crossing point of the two lines in the plot. For many applications, most users of a blocking procedure will prefer gaining recall to gaining precision as long as they can bear the cost of additional comparisons. Therefore a lower value for t will be preferred. Of course, if no further gain in recall is expected, lowering t further makes no sense. Using the plot as guideline, suitable similarity thresholds range from .55 to .6.

The choice of the recursion stop criterion s does not influence precision or recall, but affects the time needed to search the Multibit tree. Kristensen et al. [24] found $s = 6$ optimal in their application of searching molecular fingerprints. Based on this finding we tested $s = 2, 4, \ldots, 20$ and $t = .5, .6$, and .7 as similarity thresholds. Figure 7 shows that the optimal value of [24]

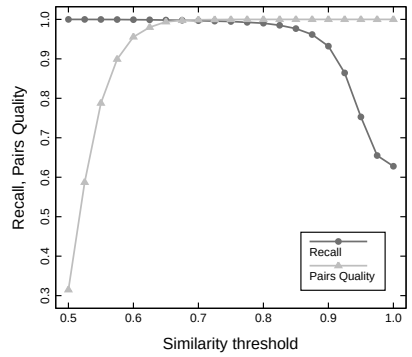

Fig. 6. Recall and Pairs Quality versus Similarity Threshold.

($s = 6$) would be good choice, but for the $Q-gram$-Fingerprints $s = 8$ seems to be better. The choice of larger s does not reduce the query time further.

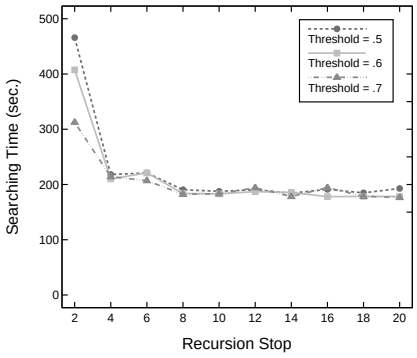

Fig. 7. Searching time for recursion stop $s = 2, 4, \ldots, 20$ at similarity thresholds $t = .5, .6$, and $.7$.

Finally, it must be decided if the partition step of the Multibit tree should be included or not. This will influence only the runnning time. Using the best performing parameters so far (trigrams with rule (c 3) and $s = 8$ for the recursion), we examined the time needed to search the Multibit tree with and without partition step for different similarity thresholds. Omitting the partition step results in lower searching times at all similarity thresholds (Figure 8). Therefore, all subsequent experiments omitted the partition step.

Summarizing the parameter selection experiments, the best choices for the Q-gram Fingerprint filtering seem to be: (a) setting the length m of the fingerprints to twice the expected number of bits set to 1, (b) use of trigrams for identifiers like names, (c) rule (c 3) for the number of hash functions, (d) a similarity threshold between .55 and .6, (e) $s = 8$ for the recursion stop, and (f) omitting the partition step when building the Multibit tree.

Scalability of Q-gram Fingerprint Filtering The scalability of Q-gram Fingerprint filtering was evaluated with test data file pairs of increasing size (Test Databases II–XI). The best performing parameters of the previous section were used; the similarity threshold was fixed at $t = .6$.

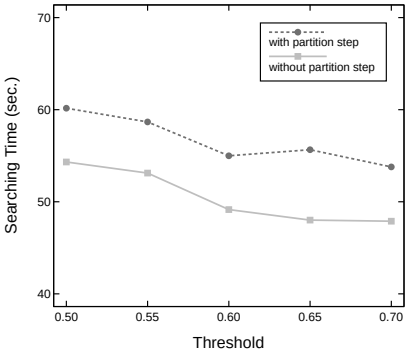

Fig. 8. Searching time with and without the partition step of the Multibit tree

Figure 9 shows the time required to build the Multibit tree for increasing file sizes. Building the tree for $2,000,000$ fingerprints took just about 20 seconds. With regard to building time, Q-gram Fingerprint filtering is clearly scalable.

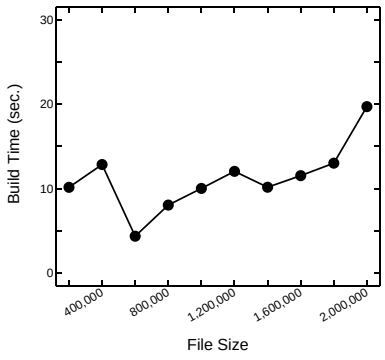

Fig. 9. Time to build the Multibit tree depending on file size

After the tree structure for the first file is built, the records of the other file are queried sequentially. Therefore, for the overall running time the time required for querying the records of the second file is more important than the time for building the tree. Figure 10 shows the average query time (1000 queries) against the size of the data file stored in the Multibit tree. The plot shows an almost linear increase in the time required to search the Multibit tree per query fingerprint, suggesting good scaling properties.

Finally, the total running time for building and querying a Multibit tree is shown in Figure 11, using the number of possible comparisons between the two files as x-axis. Again, the increase in running time is approximately linear. For many applications with up to 1 million records per file (implying nearly 500 billion possible record pairs), the total running time of 5:15 hours for Q-gram Fingerprint Filtering seems to be acceptable.

Comparison Study We compared Q-gram Fingerprint Filtering (QFB) with those three blocking methods, which performed quite well in a recent comparison study [4]: Standard Blocking (SB),

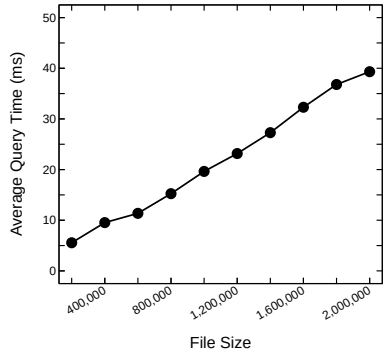

Fig. 10. Average query time for 1000 queries depending on file size

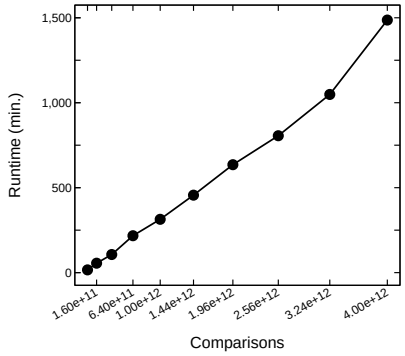

Fig. 11. Total running time depending on possible comparisons

the Sorted Neighbourhood Method (SNM), and Canopy Clustering (CC). In addition, we included DCS++, a recently proposed and well performing variant of the Duplicate Count Strategy [5].

All blocking methods were tested with Test Database XII with 100,000 records, generated in the way described in section 4.4, containing 40% records with errors. To check for the possible effect of different error rates, all experiments were repeated with the otherwise identical Test Database XIII, but which contained fewer records with errors (20%).

For QFB we adopted the best performing parameters from section 4.4. The similarity thresholds for the Multibit tree searches were $t = .55, .6, \ldots, .85, .9$. The parameters of SB, SNM, and CC were taken from previous work [4], [2]. For SB and SNM, seven blocking keys as listed in Table 3 were used. With SB we conducted 7 runs: In the first run we applied Key 1 alone, in the second run Key 1 and Key 2 successively, in the third run Keys 1 – Key 3, and so on. In each run, all unique pairs were combined to form the set of candidate pairs. For SNM the window size was fixed at 5; Keys 1–5 were used in the same manner as with SB. Again, in each run, all unique pairs were combined to form the set of candidate pairs. For CC, First Name, Last Name, Street, and Place were concatenated while inserting a blank between each consecutive attribute value. This concatenated string served as cluster variable. The canopies were formed by using the bigram similarity. As pairs of loose and tight (similarity) thresholds we used $(.5, .6)$, $(.55, .65)$, $(.6, .7)$, $\ldots$, $(.9, 1.0)$. For DCS++ we built the blocking key as described in [5] for person data. That is, we concatenated numbers 1–3 of Zip Code, Characters $1+2$ of Street Name, Characters $1+2$ of Surname, the 1st number of House Number, the 1st character of Place Name,

Table 3. Blocking keys for Standard Blocking and the Sorted Neighbourhood Method in the comparison study.

Key	Composition
1	Phonetic code of Surname + Zip Code
2	Phonetic code of First Name + Phonetic code of Place Name
3	Phonetic code of Street Name + Phonetic code of Surname
4	Zip Code + 1st character of Surname
5	1st character of Surname + 1st character of First Name + Date of Birth
6	Characters 1–3 of Surname + Characters 1–3 of First Name + Birth Month
7	Characters 1–3 of First Name + Numbers 1–3 of Zip Code + House Number

and the 1st character of First Name. As distance based classifier the mean of the bigram similarities between the First Names, the Surnames, the Street Names, and the Place Names of a record pair was used. A similarity threshold of .65 was adopted. For initial experiments, window sizes as suggested by [5] (2–1000) were used, but we found them generally too restrictive. For the experiments reported here, window sizes of $w = 1000, 2000, \ldots, 10,000$ were used.

Number of comparisons and recall for all tested blocking methods are shown in Figure 12 for Test Database XII.

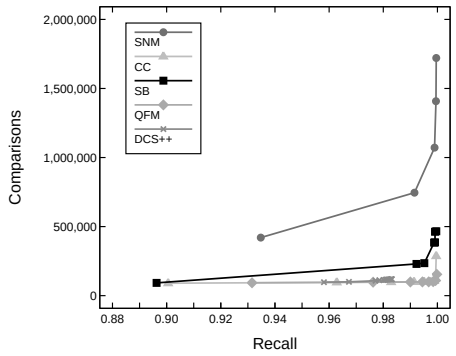

Fig. 12. Number of Comparisons versus Recall for different blocking techniques (Test Database XII)

SNM, SB, CC and QFB approach nearly perfect recall values of 1.0. However, SNM pays that by much more comparisons than SB. CC and even more QFB needed lower numbers of

comparisons for the same recall values. Due to a steep increase in running time with rising window size, we were not able to reach the same recall values with DCS++ (cf. Figure 13). Since both CC and QFB reach high recall values with small number of comparisons, both yield low running times. In contrast to this, the high recall values of SB and especially SNM are due to higher number of comparisons, resulting in higher running times. The inability of DCS++ to reach recall values as high as the other methods suggests that DCS++ seems to be unsuitable for this kind of data.

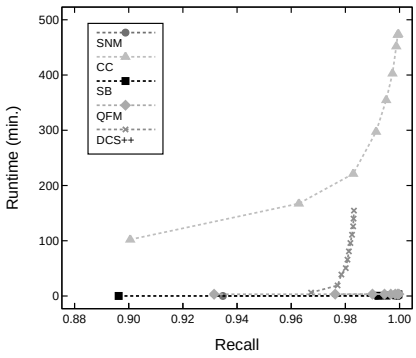

Fig. 13. Running time versus Recall for different blocking techniques (Test Database XII)

The running times of the blocking methods are shown in Figure 13. SNM, SB and QFB perform similar. All three method completed the blocking of the $100,000 \times 100,000$ record pairs in well under 5 minutes. DCS++ showed a sharp increase in running time with rising recall: For a recall of just over .990 DCS++ needed over 150 minutes. Based on this, even larger running times have to be expected for recall levels comparable to those of other methods. CC needed by far the longest running times: Nearly 500 minutes were needed for the highest recall value.

We repeated the comparison experiment with Test Database XIII which contains 20% erroneous records as compared to the 40% in Test Database XII. As can be seen from Figures 14 and 15, all findings described for Test Database XII are confirmed by the results for Test Database XIII.

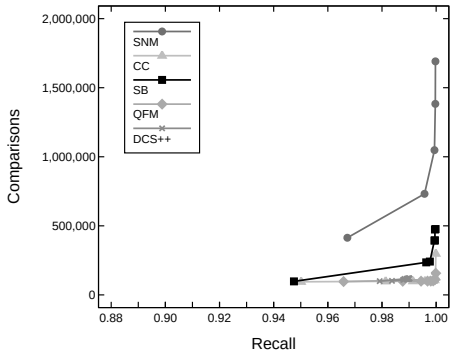

Fig. 14. Number of Comparisons versus Recall for different blocking techniques (Test Database XIII)

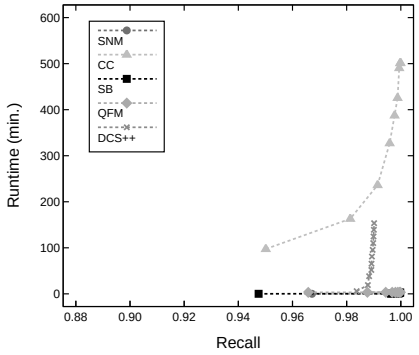

Fig. 15. Running time versus Recall for different blocking techniques (Test Database XIII)

To sum up the results, Standard Blocking and the Sorted Neighbourhood Method performed as well as QFB with respect to recall, but needed longer running times than QFB. In contrast, Canopy Clustering performed well with respect to recall, but failed in running time. DCS++ showed unacceptable running times, preventing a thorough evaluation of it recall performance. Q-gram Fingerprint Method performed best with respect to recall and nearly as well regarding running time as Standard Blocking and the Sorted Neighbourhood Method.

5 Conclusions and Future Work

In this paper we introduced Q-gram Fingerprint Filtering for record linkage. The new method first transforms records into bit vectors, the fingerprints, and then filters pairs of fingerprints using a Multibit tree [24] according to a user-defined similarity threshold. The similarity between two fingerprints approximates their q-gram similarity. Using simulated data, we explored optimal parameter settings for this method. Finally, we compared QFB with several alternative blocking methods. QFB outperformed in most situations the methods performing best in other comparison studies. Since QFB can be applied to whole records, there is no need for a previous search for appropriate blocking keys as for most alternative techniques. Therefore, QFB seems to be a useful addition to the set of available tools for record linkage.

Our future work will focus on finding ways to improve the scalability of QFB for very large databases. Currently, we examine the use of Q-gram Fingerprint filtering for privacy-preserving record linkage. Since the transformation of the records into fingerprints is similar to the generation of Cryptographic Longterm Keys [30], QFB can be applied for privacy preserving record-linkage with slightly adapted parameter settings.

Acknowledgements

This research has been supported by a grant from the German Research Foundation (DFG) to Rainer Schnell. Authors' contributions: RS suggested the use of Multibit Trees for similarity filtering in record linkage. TB and JR devised the generation of Q-gram Fingerprints from records. All algorithms and data structures were implemented by JR. TB designed the tests, TB and JR performed the experiments. TB drafted the manuscript, RS wrote the final version. The authors would like to thank Günter Törner for helpful discussions.

References

1. Ahmed Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios Verykios. Duplicate record detection: a survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.
2. Thomas N. Herzog, Fritz J. Scheuren, and William E. Winkler. *Data Quality and Record Linkage Techniques*. Springer, New York, 2007.
3. Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, Heidelberg, 2012.
4. Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555, 2012.
5. Uwe Draisbach, Felix Naumann, Sascha Szott, and Oliver Wonneberg. Adaptive windows for duplicate detection. In *28th International Conference on Data Engineering: 1–5 April 2012; Arlington, VA*, 2012.
6. Rohan Baxter, Peter Christen, and Tim Churches. A comparison of fast blocking methods for record linkage. Technical Report 03/139, CSIRO Mathematics, Informatics and Statistics, 2003.
7. Akiko Aizawa and Keizo Oyama. A fast linkage detection scheme for multi-source information integration. In *Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration: 8–9 April 2005; Tokyo*, pages 30–39, Los Alamitos, CA, 2005. IEEE.
8. de Vries, Timothy, Hui Ke, Sanjay Chawla, and Peter Christen. Robust record linkage blocking using suffix arrays. In David Cheung, Il-Yeol Song, Wesley Chu, Xiaohua Hu, and Jimmy Lin, editors, *Proceedings of the 18th ACM Conference on Information and Knowledge Management: 2–6 November 2009; Hong Kong*, pages 305–314, New York, 2009. ACM.
9. de Vries, Timothy, Hui Ke, Sanjay Chawla, and Peter Christen. Robust record linkage blocking using suffix arrays and Bloom filters. *ACM Transactions on Knowledge Discovery from Data*, 5(2):9:1–9:27, 2011.
10. Liang Jin, Chen Li, and Sharad Mehrotra. Efficient record linkage in large data sets. In *Proceedings of the 8th International Conference on Database Systems for Advanced Applications: 26–28 March 2003; Kyoto*, pages 137–146, Los Alamitos, CA, 2003. IEEE.
11. Howard B. Newcombe and James M. Kennedy. Record linkage: making maximum use of the discriminating power of identifying information. *Communications of the ACM*, 5(11):563–566, 1962.
12. Howard B. Newcombe. *Handbook of Record Linkage: Methods for Health and Statistical Studies, Administration, and Business*. Oxford University Press, Oxford, 1988.
13. Mauricio A. Hernández and Salvatore S. Stolfo. Real-world data is dirty: data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.
14. Su Yan, Dongwon Lee, Min-Yen Kan, and Lee C. Giles. Adaptive sorted neighborhood methods for efficient record linkage. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries: 17–22 June 2007; Vancouver*, pages 185–194, New York, 2007. ACM.
15. Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In Raghu Ramakrishnan, Sal Stolfo, Roberto Bayardo, and Ismail Parsa, editors, *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining: 20–23 August 2000; Boston*, pages 169–178, New York, 2000. ACM.
16. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
17. Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: a survey. *Internet Mathematics*, 1(4):485–509, 2003.
18. Esko Ukkonen. Approximate string matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, 1992.
19. Michel Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer, Berlin, 2009.
20. Richard C. Angell, George E. Freund, and Peter Willett. Automatic spelling correction using a trigram similarity measure. *Information Processing and Management*, 19(4):255–261, 1983.
21. Edward M. Riseman and Allen R. Hanson. A contextual postprocessing system for error correction using binary n-grams. *IEEE Transactions on Computers*, C-23(5):480–493, 1974.

22. Adam Kirsch and Michael Mitzenmacher. Less hashing same performance: building a better bloom filter. In Yossi Azar and Thomas Erlebach, editors, *Algorithms-ESA 2006. Proceedings of the 14th Annual European Symposium: 11-13 September 2006; Zürich, Switzerland*, pages 456–467, Berlin, 2006. Springer.
23. Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Hmac: keyed-hashing for message authentication. Internet RFC 2104, 2009.04.10 1997.
24. Thomas G. Kristensen, Jesper Nielsen, and Christian N. S. Pedersen. A tree-based method for the rapid screening of chemical fingerprints. *Algorithms for Molecular Biology*, 5(9), 2010.
25. Ramzi Nasr, Thomas G. Kristensen, and Pierre Baldi. Tree and hashing data structures to speed up chemical searches: analysis and experiments. *Molecular Informatics*, 30(9):791–800, 2011.
26. Tobias Bachteler and Jörg Reiher. Tdgen: A test data generator for evaluating record linkage methods. Technical Report wp-grlc-2012-01, German Record Linkage Center, 2012.
27. Peter Christen and Angus Pudjijono. Accurate synthetic generation of realistic personal information. In Thanaruk Theeramunkong, Boonserm Kijsirikul, Nick Cercone, and Tu-Bao Ho, editors, *Advances in Knowledge Discovery and Data Mining. Proceedings of the 13th Pacific-Asia Conference: 27–30 April 2009; Bangkok*, pages 507–514, Berlin, 2009. Springer.
28. Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. KNIME: The Konstanz Information Miner. In Christine Preisach, Hans Burkhardt, Lars Schmidt-Thieme, and Reinhold Decker, editors, *Data Analysis, Machine Learning and Applications. Proceedings of the 31st Annual Conference of the Gesellschaft für Klassifikation: 7–9 March 2007; Freiburg*, pages 319–326, Berlin, 2008. Springer.
29. Matthew A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 Census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.
30. Rainer Schnell, Tobias Bachteler, and Jörg Reiher. A novel error-tolerant anonymous linking code. Working Paper WP-GRLC-2011-02, German Record Linkage Center, 2011.

IMPRINT

Publisher

German Record-Linkage Center
Regensburger Str. 104
D-90478 Nuremberg

Template layout

Christine Weidmann