Arabian Journal for Science and Engineering (2018) 43:7409–7420
https://doi.org/10.1007/s13369-018-3105-6

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

Improved Duplicate Record Detection Using ASCII Code Q-gram Indexing Technique

Mayada A. Elziky[1] · Dina M. Ibrahim[1] · Amany M. Sarhan[1]

Received: 29 December 2016 / Accepted: 4 February 2018 / Published online: 15 February 2018

Abstract
With the aim of reducing duplicate records in databases, duplicate record detection (DRD) ensures the integrity of data. Its role is to identify records signifying same entities either in the same or in different compared to database. A diversity of indexing techniques has been proposed to support DRD. Q-gram is one of the common techniques used to index databases. This paper introduces modification to the Q-gram indexing technique. Such modification participates in improving the performance of the duplicate detection process and in reducing the time and number of comparisons. In the proposed work, in order to make the back-end computations easier, Q-gram strings are alternatively converted into numeric values using their corresponding ASCII code. Based on these numeric values, the indexing will decrease the complexity of Q-gram comparisons and speed up the DRD process as a whole. Unlike the existing approaches, the proposed technique is easier in implementation and requires less memory space. Two other variations of the proposed technique are introduced in this paper to decrease the matching process time; the first uses a range for matching, while the second sorts words alphabetically inside blocks. According to experimental results, the three proposed techniques perform much faster and are almost as accurate as the current Q-gram technique, meaning that they can be used in large-sized databases DRD.

Keywords Duplicate record detection · Q-gram · Indexing technique · BKV · ASCII code

1 Introduction

Most agencies and companies produce and handle huge amounts of data. Hence, new techniques are needed to make the processing of large-sized databases much easier and more efficient. To ensure the reliability of such techniques, firstly, data must be free from any errors, such as duplication. To achieve this goal, records pointing to same objects need to be matched from databases. Enforcing data integrity guarantees the accuracy and coherence of data. According to this, DRD plays a significant role in enhancing data integrity [1]. As multiple source information may be integrated to make detailed data analysis easier, duplication problem may occur. Currently, a plethora of entities, such as tax agencies, banks, mobile network operators, universities and medical research centers, use DRD to detect and, thus, reduce duplicate records in real-world databases. Furthermore, DRD is highly helpful in other fields such as fraud detection and money laundering [2].

The major problem in the detection of duplicate records is the number of comparisons to be made between records. To solve such problem, indexing was introduced to sort or gather database rows according to one or more fields. Database indexes are methods used to rapidly reach data items in databases, without having to search every row in a database table every time it is accessed [1,2]. By using indexing, rows can be combined in groups (called blocks), according to a blocking key value (BKV). BKV can be either a single attribute or a combination of two or more attributes. As such, the number of comparisons can be reduced, because they are now limited to the rows found in a block.

Moreover, indexing is used in identity matching, where it is important to enforce personality identification efficiency of matching and expandability for large datasets [3]. The step of indexing decreases the number of candidate identity

✉ Amany M. Sarhan
amany_sarhan@f-eng.tanta.edu.eg

Mayada A. Elziky
mayada.elziky@f-eng.tanta.edu.eg

Dina M. Ibrahim
dina.mahmoud@f-eng.tanta.edu.eg

[1] Department of Computers and Control Engineering, Faculty of Engineering, Tanta University, Tanta, Egypt

pairs by explicitly removing non-matching pairs, before the decision-making model step, which makes explicit comparisons. Such perspective is basically used in record linkage and de-duplication research. However, de-duplication research is similar to identity matching research, since they both include searching duplicate records in a database.

Many indexing techniques have been presented in the literature to improve DRD. Some of such techniques are: traditional blocking [4], sorted neighborhood indexing [5], suffix array-based indexing [6], canopy clustering [7] and Q-gram-based indexing [8].

Q-gram indexing is considered the most popular and accurate indexing technique. Q-gram indexing is based on Q-grams [8,9] which are short sub-strings of length q of a given string. A sliding window of length q is performed on the characters of the given string to obtain Q-gram sub-strings. This technique is used to index databases such that records with similar, not just the same, BKV are inserted in the same block. Within this block, each string will be further compared using a matching function to find only those with the closest match.

In addition to DRD, Q-gram has been used in many other fields. In [9], Q-gram fingerprinting blocking technique was used for maintaining high-quality linkage in reasonable time. In [10], a birthmarking approach using Q-grams is presented. Using this technique, designing patterns regarding existing circuits can be captured and used to suggest not only similar and reusable designs, but functional blocks throughout the design phase, with little effort from the user. In [11], because of the huge number and the dimensions' sparseness of the scientific research text set, a new similarity search algorithm for the scientific research text set is proposed, which is based on the weights of the Q-gram. The algorithm can greatly reduce the number of dimensions, and then quickly find the similar text.

On the other hand, the increase in the database size imposes a challenge to the detection process since more time is required for large database size. To deal with this problem, in this paper, we concentrate on providing more accurate, faster and efficient indexing technique to be used in the DRD matching or in any other database matching processes, given that we can improve the performance of the lookup step by using better indexing approach, particularly in large size datasets [12,13]. With the view of achieving such goal, we introduce a modification to Q-gram indexing, by converting the Q-gram strings into numeric values via substituting each string with its corresponding ASCII code and performing indexing on such numbers. The aim of such modification is reducing the back-end computations, by reducing the complexity of the comparison step and, thus, speeding up the DRD process.

The remaining of this paper is organized as follows: Sect. 2 gives an overview of the DRD process and Q-gram; Sect. 3 presents the proposed indexing technique; Sect. 4 includes experiments and results; Sect. 5 discusses and explains the obtained results; finally, Sect. 6 concludes the paper and suggests future work.

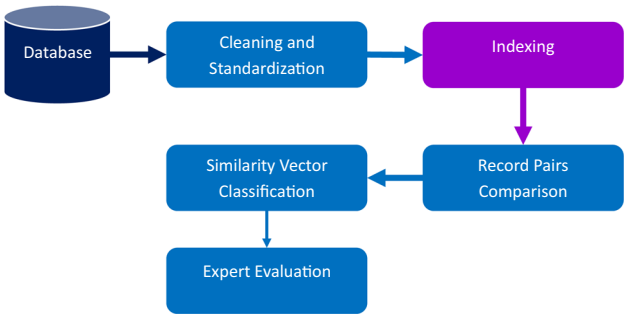

Fig. 1 The main steps of DRD

2 Background and Related Work

The majority of real-world data is uncertain and contains noisy, imperfect and misformatted information. To clean such data before being processed, DRD can be used. DRD is a technique used to detect pairs of records in one or more datasets that point to same entity. Duplicate detection methods usually depend on string similarity to differentiate between matching (i.e., duplicate) and non-matching (i.e., non-duplicate) record fields. In addition, duplicate record methods depend on record similarity for combining similarity estimates from individual string fields [14].

2.1 Main Steps of DRD

There are six main steps of DRD as depicted in Fig. 1. The role of each step is summarized as follows [3].

2.1.1 Data Cleaning and Standardization Step

The main and first step in any record DRD approach is data cleaning and standardization [15]. It was proven that the existence of poor quality data may limit the effectiveness of record linkage. The transformation of input data into a well-defined form is the main function of data cleaning and standardization process. It also helps solving any inconsistencies in the way information is represented and encoded by different data sources [16].

Data cleaning and standardization is the process of detecting and correcting (or deleting) messy and inaccurate records from a record set, table or database. This term refers to identifying incomplete (or incorrect, inaccurate, irrelevant, etc.) parts of these data and replacing, modifying or omitting this unclear or poor data. Data selection may be carried out

interactively with data wrangling tools, or as data processing through programming [17].

In the DRD framework, cleaning and standardization can be performed either on character level or on string level. As for the character level, spaces are trailed and removed, or specific characters are replaced with a set of equivalent values. Composite names and words often face the problem of typographical differences, which will lead to a negative false comparison. They have to be converted to specific format to guarantee correct detection between records. As for the string level, rules can be added to remove titles, abbreviations or prefixes, such as Eng., Mr., prof., Dr., … etc. [18]. This step is flexible and language-based in the sense that any additional rules can be set and applied to normalize the data.

2.1.2 Indexing/Blocking Step

The goal of the indexing step is to decrease the great number of possible comparisons by detecting all non-matching record pairs. In the comparing step, these candidate records pairs are compared using a suitable comparison function according to the content of the field attributes. Many indexing techniques were introduced in the literature. Some of them will be listed below.

Traditional blocking has long been used in record linkage [5] where all records having the same blocking key value (BKV) are put in the same block such that each record is put only in one block. Afterward, these records within the same block are compared with each other to detect their similarity. The main weakness of traditional blocking is the mistakes and variances in data in record fields which generate BKVs that cause records to be assigned to improper blocks. This disadvantage can be controlled by using several blocking key definitions that are based on various record fields; or the various encodings which have already been applied on the same record fields [19].

Sorted neighborhood indexing came out to use in the mid-1990s [20]. Its main idea is to sort the database(s) in accordance with the BKVs, and to consecutively move a window of a fixed number of records w ($w > 1$) over the sorted values. To produce the candidate record pairs, the records available in a current window are used. Yet, the main disadvantage of this method is the sensitivity of sorting the BKVs to mistakes and variances in the first few positions of values. For instance, given two names: 'Careem' and 'Kareem,' if they were used as BKVs, they seemed far away in the sorted array; however, they must be inserted into the same window. This shortcoming can be conquered by using multiple blocking key definitions based on various record fields, or reversed field values [21].

Suffix array-based indexing has lately been proposed as an effective autonomous domain approach to multi-source information incorporation [4]. Its fundamental notion is to insert the BKVs and their suffixes into a suffix array based on inverted index. A suffix array includes alphabetically arranged strings [6]. In this indexing technique, only suffixes that down to a minimum length are inserted into the suffix array. The main disadvantage of suffix-based indexing, that will put records into different blocks, is mistakes and variances at the end of BKVs, and thus missing true matches. To avoid this disadvantage, a modification of the suffix generation process, that does not only produce the true suffixes of BKVs but also all the sub-string down to minimum lengths in a sliding window shape, is proposed in [21].

Canopy clustering is an indexing technique that uses a computationally inexpensive clustering approach to produce high-dimensional overlapping clusters. Using these overlapped clusters, blocks of candidate record pairs can be generated [4,22]. The calculation of the similarities between BKVs is computed using standard similarity functions such as Jaccard or TF-IDF/Cosine [7]. These similarity functions work on tokens, which can be characters, Q-grams or words [7,23]. However, the drawback of this approach, which is the same as of the sorted neighborhood technique, is that the generated clusters of BKVs that are so common (like the surnames 'Al-Khouly' or 'Al-Bana'), might not include all records with these BKVs, and therefore, the matching between these records cannot be correctly predicted. Another work in [23] has proposed to use BKVs that have a sequence of several record fields and a large number of different values.

Finally, Q-gram indexing [24], which is the main topic of this paper, is going to be discussed in detail in Sect. 2.2.

2.1.3 Record Pair Comparison, Similarity Vector Classification and Expert Evaluation Steps

In the comparison step, string comparisons are performed in fields with string content such as name, faculty or country. Typographical variations should be taken into consideration to overcome such problem [25]. Other comparison functions are used for fields containing other types of data such as date or numeric data types [12]. Similarity values are obtained via comparing the corresponding fields with other record pairs. These values are then stored in a vector that represents the similarity between pairs.

As previously discussed in this paper, the insufficiency of distinct identifiers throughout integrated data sources has proven the necessity for carrying out record comparisons using string fields. A lot of methods have already been evolved to do string matching. Considering the wide range of typographical differences, each method performs well with a particular sort of typographical difference/error [13].

The most commonly used techniques for matching string fields rely on either character-based similarity metrics, token-based similarity metrics or phonetics similarity metrics [26]. The character-based similarity metrics, used to tackle typo-

graphical variations/errors, include Jaro–Winkler distance, Levenshtein edit distance, longest common subsequence and Hamming distance [26,27].

Making use of similarity values, the next step in the DRD process is to separate the compared selected record pairs into duplicates, non-duplicates and possible duplicates, relying on the decision model used [26]. Record pairs excluded in the indexing step are classified as non-matches and will not be compared. In the framework, a range is specified for a match between {60%:100%}, non-match between {0%:40%} and possible match between {40%:60%}. The final step in the process is assessing the quality of the DRD results [26]. This can be performed through the help of an expert.

2.2 Q-gram Indexing Technique

Q-gram indexing technique is associated with the problem of the long time required to perform indexing, due to the large number of BKVs generated in the inverted index to represent records [9–11]. However, Q-gram indexing technique is the most reliable one, as the records in this technique are allocated to multiple index keys to escalate efficiency. It converts a BKV into Q-gram sub-lists which are merged to create index key values.

The Q-grams are simply sub-strings of length q; the concept was presented by Shannon [28]. They have been utilized in many variances, e.g., in different spelling correction methods. The text is first preprocessed maintaining a static dictionary to make subsequent searches for 'correct' words faster. It is identified by Ukkonen [29] as brief character sub-strings of length q of the database strings. Moreover, Q-grams, including unigrams, bigrams and trigrams, used different ways for spelling correction and text recognition that explicated by Kukich [30].

Gravno et al. [31] continued his work to extend the capabilities of handling spelling errors by using Q-grams instead of words as tokens. By doing so, a spelling error has a slight effect on the set of ordinary familiar Q-grams of two strings. Therefore, the two strings 'Treaties' Weapons' and 'Weapon Treaties' have high similarity under this metric, in spite of the block shift and the spelling errors in both phrases. This measure handles the addition and omission of words delicately. The string 'Treaties' Weapons' matches with high similarity the string 'International Weapon Treaties' because the Q-grams of the word 'International' appear so much in the relation and has low weight.

Elmagarmid defined Q-gram as a subsequence of q points [13]. These points may be whole words, syllables or letters of phonemes. A 'unigram' refers to a Q-gram of size 1, 'bigram' to size 2, 'trigram' to size 3, and size 4 or more simply called 'Q-gram'.

On a Q-gram metric of similarities, tokens are not identified in relation to characters (white spaces and punctuations), but they are turned into smaller tokens based on the size q which are called Q-grams [32]. Furthermore, these tokens usually overlap (one character appears in many tokens) q tokens in particular. In order to reproduce a size q, a window must be sled over the string in order to be tokenized (made into tokens). In order to gather q tokens, they present a special non-alphabetical character (such as # or _) and pad the string with the alphabets. The main advantage of this approach is its ability to conceal errors and variations in blocking key values (BKVs). Truly matching objects are most likely grouped into the same block which will lead to a correct matching.

Let r be a string. Any sub-string of length q (> 0) in r is called a Q-gram. A positional Q-gram comprises a Q-gram and its position in r. For example, the 3-grams of the string 'amany' are {'ama,' 'man,' 'any'} and the positional 3-grams of 'amany' are {(ama, 1), (man, 2), (any, 3)}.

Given BKVs are strings; the basic idea is to produce many versions for each BKV using Q-grams, and to interject the record in more than one block. One of the basic demerits of this step is its liability to many mistakes and variances. Moreover, this step might take so much time. A list of Q-grams is generated from each BKV where these sub-lists are combined to a particular minimum length, specified by a threshold t, selected by users in a previous step. If a BKV contains k Q-grams, then sub-list combinations, starting from k to a minimum length of l, are created [21].

Table 1 illustrates examples of applying the Q-gram indexing for three records, where $q = 2$ (bigrams), given a threshold value $t = 0.8$. In the first record (R1), the BKV 'Ahmed' produced four ($k = 4$) bigrams: 'ah,' 'hm,' 'me,' 'ed' (presuming

Table 1 Q-gram based indexing used as BKVs (first name), ($q = 2$)

Identifiers	BKVs (first name)	Bigram sub-lists	Index key values
$R1$	Ahmed	[ah, hm, me, ed], [hm, me, ed], [ah, me, ed], [ah, hm, ed], [ah, hm, me]	**ahhmmeed**, hmmeed, ahmeed, ahhmed, ahhmme
$R2$	Ahmedy	[ah, hm, me, ed, dy], [hm, me, ed, dy], [ah, me, ed, dy], [ah, hm, ed, dy], [ah, hm, me, dy], [ah, hm, me, ed]	ahhmmeeddy, hmmeeddy, ahmeeddy, ahhmeddy, ahhmmed, **ahhmmeed**
$R3$	Ahmede	[ah, hm, me, ed, de], [hm, me, ed, de], [ah, me, ed, de], [ah, hm, ed, de], [ah, hm, me, de], [ah, hm, me, ed]	ahhmmeedde, hmmeedde, ahmeedde, ahhmedde, ahhmmede, **ahhmmeed**

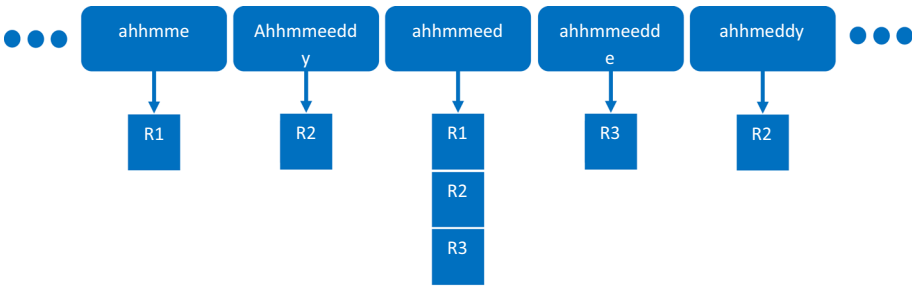

Fig. 2 The resulting inverted index lists (blocks)

that all letters have already been turned into lowercase). From the table, we can compute the shortest sub-lists length $\boldsymbol{l}$ for this value to be $\boldsymbol{l} = 3$. Therefore, four sub-lists are produced each containing three bigrams for BKV 'Ahmed' as: [hm, me, ed], [ah, me, ed], [ah, hm, ed], and [ah, hm, me]. These sub-lists were produced by removing one of the four bigrams in the original list. The key values used in the inverted index are produced from these sub-lists after being converted to strings, as shown in Fig. 2 [21].

Definition 1 (*Sub-lists*) Let M be a given BKV. We can generate a list of Q-grams from M. A sub-list of M is a combination of the generated Q-gram elements. The sub-list combinations continue to a certain minimum length $\boldsymbol{t}$ ($\boldsymbol{t} \leq 1$), which is a user selection parameter. If $\boldsymbol{k}$ is the number of bigrams in BKVs, all sub-list combinations down to minimum length of $\boldsymbol{l} = \max(1, [\boldsymbol{k} \times \boldsymbol{t}])$ will be created ([…] corresponds to approximating the float number to the lower integer). Let $\boldsymbol{S}$ be a sub-list of Q-grams for a given string. The number of sub-lists can be computed as [21]:

$$S = \sum_{i=l}^{k} \binom{k}{i}. \tag{1}$$

2.3 Jaro Similarity Distance

Jaro similarity distance is a type of edit distance approaches used for measuring similarity between two strings. It was used in many applications such as matching and search. Higher Jaro distances between investigated strings indicate higher possibility of being two similar strings. However, it is only suitable for comparing short strings (e.g., names). (0) distance means no similarity, while (1) distance means an exact match. In the proposed framework, the Jaro string matching function is chosen because it takes into consideration the number of matched characters and the number of transportations required, regardless of the length of the compared strings. Some of the other string matching functions cannot be used due to the nature of the DRD application [29,32].

Definition 2 (*Jaro distance*) Let $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ be two given strings. Jaro distance ($\boldsymbol{d}_j$) between these two strings is computed as follows: if there are no matched characters, then $\boldsymbol{d}_j(\boldsymbol{s}_1, \boldsymbol{s}_2) = 0$, otherwise, when $\boldsymbol{m}$ is the number of matched characters and $\boldsymbol{t}$ is the number of transportations, it is computed as [29,32]:

$$d_j\,(s1 \circ s2) = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s1|} + \frac{m}{|s2|} + \frac{m-t/2}{m}\right) & \text{otherwise} \end{cases} \tag{2}$$

This requires $O(|s_1| \cdot |s_2|)$ time for comparing two strings of length $|s_1|$ and $|s_2|$.

3 The Proposed ASCII Code Q-gram Indexing Technique (ACQIT)

The proposed indexing technique in this paper has the objective of enhancing the performance of DRD process by reducing the number of compared record pairs preserving the accuracy. With the aim of boosting its capability to index strings, we propose an improvement of the Q-gram indexing technique. The goal behind the use of Q-grams is that when two strings s_1 and s_2 have a small edit distance, they have many Q-grams in common, while the use of positional Q-grams involves the comparison of the positions of matching Q-grams within a certain distance. This incurs high complexity and requires long time. However, we found that if it is possible to represent the Q-gram string as a number, this will reveal the possibility of two strings having the same content if they have the same number without requiring to match each of the Q-grams sub-strings at different positions.

The basic idea of the proposed work is to quantify a Q-gram string into an integer number, which is later used to index strings. Using the quantified numeric value enables the indexing phase to quickly detect candidate records that have numeric similarity, thus reducing the overall execution time of the whole process. This technique can reduce the number of comparisons and the amount of time required to perform comparisons by excluding non-matches in an efficient manner. In this way, the complexity of the indexing process is reduced while preserving its accuracy, as will be shown later.

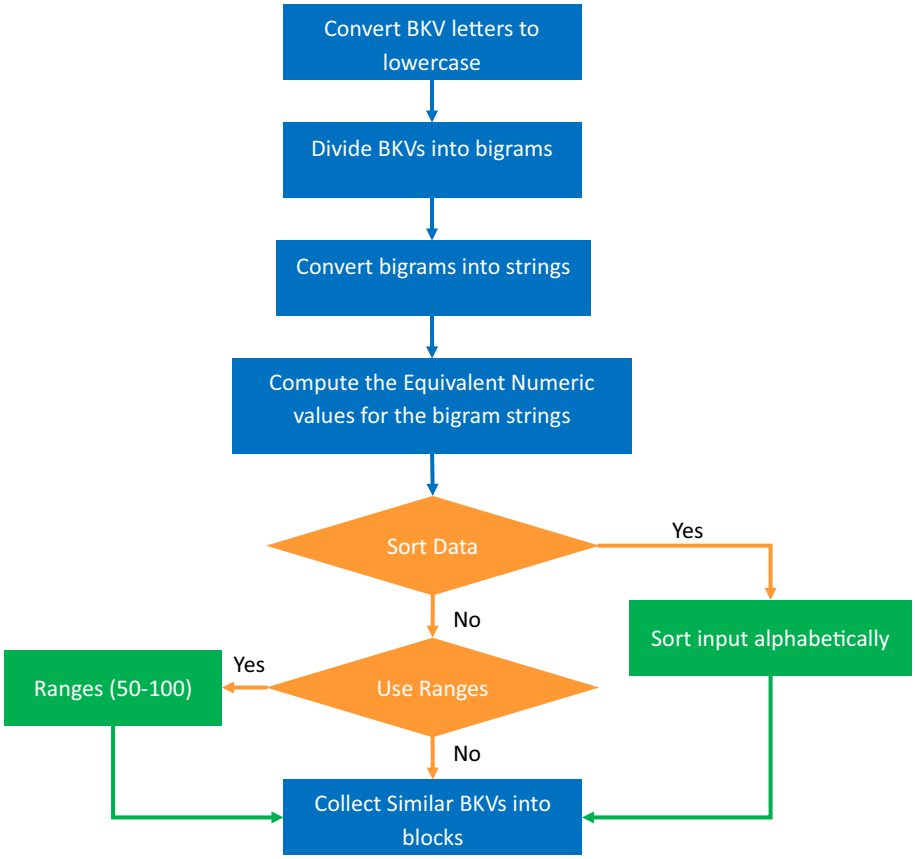

Fig. 3 Steps of the proposed indexing technique

String	*Bigrams*			
Sameh	sa,	am,	me,	eh,
	1st bigram	2nd bigram	3rd bigram	4th bigram

Fig. 4 Example of a string (Sameh) divided into bigrams ($q = 2$)

The proposed ASCII code Q-gram indexing technique (ACQIT) has the following steps, as depicted in Fig. 3:

1. Convert all the letters of the BKV string into lowercase letters to remove the effect of uppercase letters in the conversion to numeric value.
2. Adjust the Q-gram to 2, i.e., the BKV is divided into bigrams. An example is given in Fig. 4 where we divide the string (Sameh) into bigrams.
3. Convert these bigrams back into strings.
4. Compute the equivalent quantified numbers for the bigram strings through multiplying each letter position or sequence in the BKV by its ASCII code value shown in Table 2. This number will be used later to determine the similarity between BKVs.

Table 2 Characters ASCII code

ASCII code	Alphabet	ASCII code	Alphabet	ASCII code	Alphabet
97	a	107	k	117	u
98	b	108	l	118	v
99	c	109	m	119	w
100	d	110	n	120	x
101	e	111	o	121	y
102	f	112	p	122	z
103	g	113	q		
104	h	114	r		
105	i	115	s		
106	j	116	t		

Definition 3 (*String numeric equivalent*) Let us have a string (***S***) containing ***N*** characters, ***ASCII*** is the ASCII code value for a character, ***i*** is character position in the word, and S_i refers to a character in the string, the bigram string numeric equivalent (***String-eqv***) is obtained by using the following equation:

$$\text{String-eqv} = \text{ASCII}(S_1) + \sum_{i=2}^{N-1} 2 * i * \text{ASCII}\,(S_i) + N^* \text{ASCII}(S_N) \quad (3)$$

5. Finally, similar numeric value BKVs are grouped into blocks on which the matching process will be made.

The proposed technique uses the exact match to group the BKVs. This, however, gives no possibility for any misspelling or transpositions in the string entry session. Two variations of this technique are presented to solve this drawback.

- The first variation of the proposed technique considers a range for similarity inside blocks. The user can set this value according to the environment under consideration; however, we have experimentally found that the ranges 50 and 100 are considered fair differences between strings to cover error possibilities in the data entry session. We call the first variation technique ACQIT with a range.
- The second variation sorts BKVs alphabetically inside blocks to reduce the matched pairs. We call this approach ACQIT Alphabet. This idea can be combined with the exact match technique (ACQIT) or with ACQIT with a range technique.

The pseudocode of the proposed ACQIT is shown in Fig. 5. The input dataset (D) contains the records to be checked for duplication. The output of this algorithm will be a second dataset (R) in which we will store the generated pairs. In lines $\{1, 2\}$, for each string in a dataset (D), all letters are converted to lowercase letters. In lines $\{3 \ldots 5\}$, the equivalent numeric value of the ASCII code for the bigram strings is computed. In lines $\{6 \ldots 9\}$, we apply the alphabetical sort if we wanted dataset (D) to be sorted. Otherwise, ranges between (0, 50) or (0, 100) are applied. In lines {10, 11}, each row (r) in dataset (D) is compared with all other rows having same values or same range according to the used approach. In lines $\{12 \ldots 16\}$, if the numeric value of the ASCII code of word (r) is equal to that of word ($r + 1$); Jaro distance is applied and r and $r + 1$ are inserted in the generated pairs (R).

The aforementioned process is demonstrated by an example in Fig. 6. In the figure, we have three strings to match: **Amany**, **Ayman** and **Yuomna**. We begin by converting the letters into lowercase letters. Then, each string is divided into bigrams and reverted back to a string. Then, the strings are converted into equivalent numbers, which can be compared easily, using Eq. (3) as depicted in Fig. 6.

Using these numbers, the comparison shows that the three strings are not identical and there is a slight difference between **Amany** and **Ayman** (2600–2561 = 39), but **Yuomna** is very remote from both; as the numeric difference between them is large. **Amany** and **Ayman** could be the same record if a data entry error occurred.

In Fig. 6, another example of two other strings is given; **Omnia** and **Omnih**. These two strings may refer to the same record. After performing the steps of the proposed technique, the difference between their equivalent values is only 35, due to the variation in writing the same name. This is an acceptable variance between the two strings, and therefore, we should tolerate it in the proposed technique to consider them

Fig. 5 The steps of ACQIT

Input: Dataset D for all the records sets
Output: Dataset R for all generated pairs

```
1.  For each Word (w) in
2.  For each Word (w) in D do
3.      For each Character (ch) in (w) do
4.          Sum(ASCII(ch))
5.          If (D, sorted) then
6.              Sort(D,Alphabetical)
7.          Else
8.              Use range(D, range between (0,50), range between (0,100))
9.         End If
10. For each record (r) in D do
        For each
11.         If (ASCII(word in r) == ASCII(word in r+1)) then
12.             If (r_name = r+1_name) then
13.                 Jaro(r, r+1)
14.                 Insert(r, r+1) into R
15.             End If
16.         End If
```

Record	Name
R1	Amany
R2	Ayman
R3	Yuomna
R4	Omina
R5	Omnih

(a)

(b)

Record	Bigram sub-list	String	String Weight	Result
R1	am, ma, an, ny	ammaanny	1*97 + 2*2*109 + 2*3*97 + 2*4*110 + 2*5*121 = 97 + 2*218 + 2*291 + 2*440 + 605	2,600
R2	ay, ym, ma, an	ayymmaan	1*97 + 2*2*121 + 2*3*109 + 2*4*97 + 5*110 = 97 + 2*242 + 2*327 + 2*388 + 550	2,561
R3	yu, uo, om, mn, na	yuuoommnna	1*121 + 2*2*117 + 2*3*111 + 2*4*109 + 2*5*110 + 6*97 = 121 + 2*234 + 2*333 + 2*436 + 2*550 + 582	3,809
R4	om, mn, ni, ia	ommnniia	1*111 + 2*2*109 + 2*3*110 + 2*4*105 + 5*97 = 111 + 436 + 660 +840 +485	2,532
R5	om, mn, ni, ih	ommnniih	1*111 + 2*2*109 + 2*3*110 + 2*4*105 + 5*104 = 111 + 436 + 660 + 840 +520	2,567

Fig. 6 Example of applying ACQIT

as possible matching records. The exact matching technique (ACQIT) considers that only equal BKVs strings give exact matching. Because of the various ways of writing names; as names actually have no fixed ways of writing, as well as unintentional keyboard mistakes or transpositions, we set a range that tolerates and regards acceptable differences between strings as matches.

The first variation of the proposed technique tries to solve such problem. That is to say, if a string numeric value equals 2000 and another string numeric value equals 2050, we consider them a possible matching pair. In this case, the range is adjusted to 50; i.e., consider them a probable match if there is a slight mistake in writing the name. Meanwhile, if the range is taken to be 100, this means there is a larger error in writing, but still we consider them a probable match but with a less probability than the smaller range. We have tested several ranges to compute the acceptable range in order to make our technique faster in performing the comparisons. In the proposed technique, we only use 50 and 100 as ranges to represent different matching degrees. However, we found out that smaller ranges might miss possible matches and bigger ranges result in more comparisons. Therefore, a trade-off must be made between accuracy and time. This value can, however, be a user's choice parameter.

Figure 7 clarifies the above discussion; the exact match technique for '**Omnia-Omnih'** strings, (Fig. 7a), has produced two different blocks and missed a possible match between the two strings. On applying the proposed technique with a range on the same example; taking the range to be 50, (Fig. 7b), the two names are put in the same block and thus can be compared based on their characters in the next step.

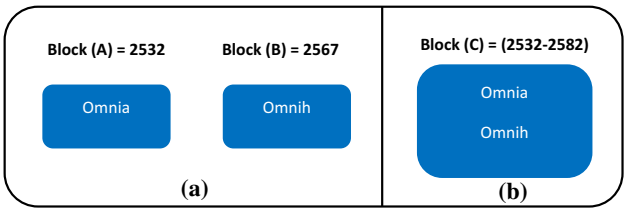

Fig. 7 ACQIT. **a** Without and **b** with a range

In the second variation of the proposed technique, we also divide each block into a number of sub-blocks which are then sorted alphabetically within the sub-block before starting records comparison so that we can reduce the effort exerted and the time consumed in comparisons. Sorting the data alphabetically inside the block helps realizing if pairs are matching or non-matching much easier, as shown in Fig. 8.

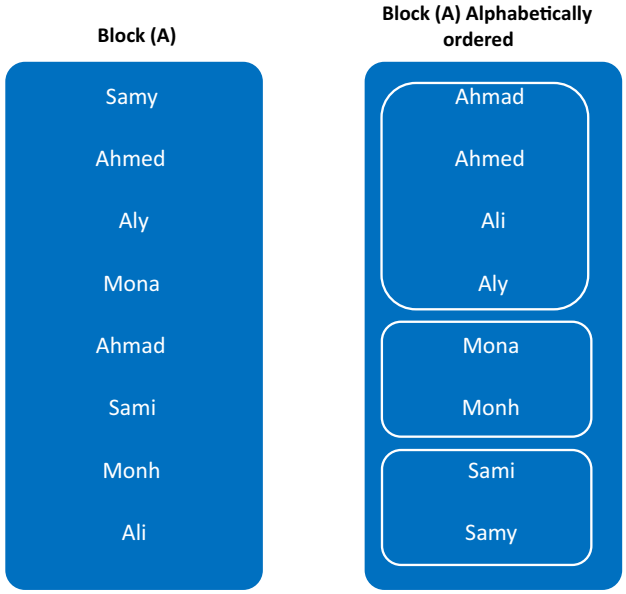

Fig. 8 ACQIT with Alphabet

4 Experiments and Results

Our simulation results were implemented on a device that has the following system configuration: Processor Intel (R) Core^{TM} 2 Duo, CPU T6600 @ 2.23 GHz, installed memory (RAM) 3.00GB, and the code were built using Visual Studio C# 2010. The performance of the proposed technique was evaluated according to two different aspects: the time consumed and the accuracy of the results. The carrying out performance of the proposed indexing technique is compared with Q-gram indexing technique to ensure that the decrease in computational time has not affected the accuracy of DRD process. In this experiment, which runs Q-gram, ACQIT, ACQIT with range (50), ACQIT with range (100) and ACQIT Q-gram Alphabet, we make use of a real dataset containing 10,000 records obtained from [23]. In all our experiments, q has been set to 2. The Jaro distance is used to obtain the match.

We have calculated the number of generated pairs, the number of comparisons and the time consumed by the various techniques, as shown in Table 3 and graphically depicted in Figs. 9, 10 and 11. The results clearly show that the proposed technique with its different variations has better behavior with respect to the other techniques in terms of the number of generated pairs, the number of comparisons and the time consumed. However, when the used range is high (100), the processing time decreased, but the generated pairs and the number of comparisons increased, which may affect the indexing process. Using the sorted alphabet variation requires less time, number of generated pairs and number of comparisons. This means achieving higher reduction ratio in the number of compared record pairs. This reduction results in a decreased computational time of DRD process, as shown in Fig. 11. The decrease in time gained by the proposed technique and its variations is in the range of 20–25%. This is due to the reduction in generated pairs and comparisons performed, which were about 74–92 and 82–99%, respectively.

Table 3 No. of generated pairs, no. of comparisons and time taken to perform comparison by the various techniques

	No. of generated pairs	No. of comparisons	Time taken to perform comparison
Dataset	10,000		
Q-gram	3539	2,650,382	2:18
ACQIT	276	15,530	1:57
ACQIT range (50)	605	255,665	1:54
ACQIT range (100)	856	480,380	1:47
ACQIT Alphabet	306	15,530	1:44

The quality of the results of the various indexing techniques is assessed by using precision, recall and F-measure.

Definition 4 (*Precision*) Precision is used to measure the quality of record linkage/duplicate record detection, and it is considered the statistical variance of an estimation procedure. Precision is measured as [33]:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

It represents the positive predictive value, where TP is the true positive value and FP is the false positive value.

Definition 5 (*Recall*) Recall is also used to measure the quality of record linkage/duplicate record detection. Recall (true positive rate) is measured as [33]:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5}$$

where FN is the false negative predicted value.

Definition 6 (*F-measure*) F-measure is a measure that combines precision and recall. The harmonic mean of precision and recall appears in [33], the F-measure (also called F-score) which is based on precision and recall is defined as:

$$\text{F-measure} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

These results are presented in Table 4. From these results, we can see that ACQIT with range (100) has the highest precision followed by ACQIT Alphabet and ACQIT, while the ACQIT with range (50) gives lower precision which is less than the original Q-gram. This is due the fact that the

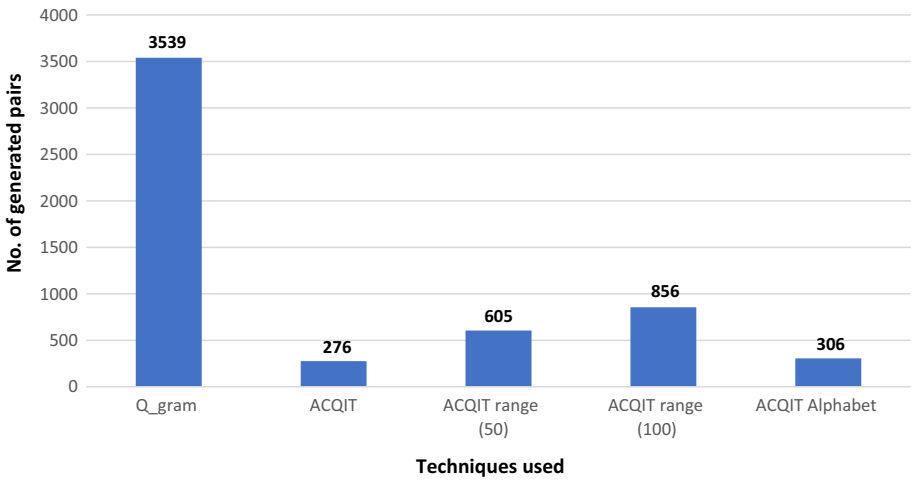

Fig. 9 Number of generated record pairs by the various techniques

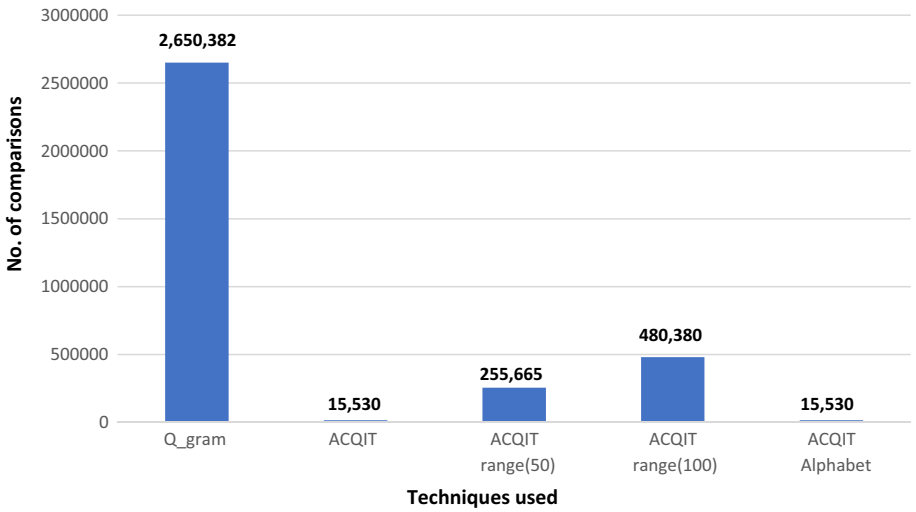

Fig. 10 Number of comparisons by the various techniques

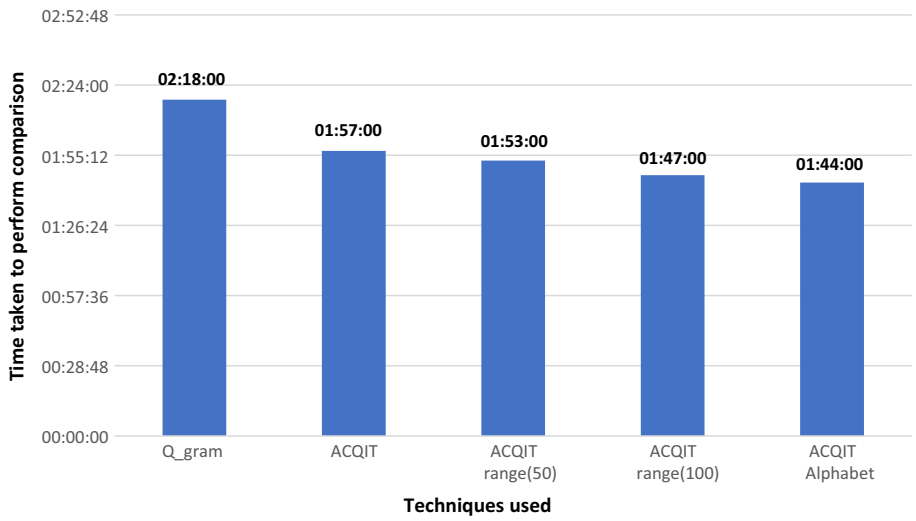

Fig. 11 Time taken to perform comparison by the various techniques (min:s:ms)

range (50) was too small to detect possible matches. However, the recall value for ACQIT with range (100) and ACQIT Alphabet reach 100%. In addition, ACQIT with range (100) has the highest F-measure followed by ACQIT Alphabet and ACQIT, while the ACQIT with range (50) gives low F-measure value which is less than the original Q-gram. In

Table 4 Quality metrics comparison for the various techniques

Technique used	True positives (TP)	True negatives (TN)	False positives (FP)	False negatives (FN)	Precision (%)	Recall (%)	F-measure (%)
Q-gram	0.82	1	0.18	0	82	100	90
ACQIT	0.70	0.91	0.30	0.09	86	96	91
ACQIT range (50)	0.77	0.98	0.23	0.02	77	97.5	86
ACQIT range (100)	0.88	1	0.12	0	88	100	94
ACQIT Alphabet	0.87	1	0.13	0	87	100	93

general, all the variations of the proposed technique, except ACQIT with range (50), have better precision and F-measure than the Q-gram and has the same recall as the Q-gram.

5 Discussion of Results

As previously mentioned, indexing goal is to locate data items within a database in an easier manner rather than having to check each possible item individually. Reducing the number of multiple data item matches, we can reduce the search space of the matching process. The number of matching operations in the worst case is $O(N)$ where N is the number of database rows. As seen from the previous results, the proposed techniques outperform the Q-gram technique in precision and F-measure parameters and equal to their recall values. This is due to the indexing being based on numbers instead of strings and because indexing through numbers always gives more accurate results as proven in the literature. Thus, strings having equal equivalent numbers are probably equal in their string values than those with different numbers. In addition to that, unlike comparing strings, comparing numbers takes little time which will reduce the time required for comparisons and facilitate using such technique for comparing large data sizes.

ACQIT with large range technique (for example, 100) is best used when high precision is required and database sizes are large. It has the advantage of requiring less time than the other techniques followed by ACQIT Alphabet and ACQIT techniques.

6 Conclusions and Future Work

DRD is extensively used in huge businesses, banks, government and non-government organizations, to tackle people's accounts, information, etc. Being highly important, it may be used everyday. We need to improve its performance and efficiency to avoid duplicate records, which, in turn, may affect people's activities. This paper handles this notion by introducing a new idea of Q-gram indexing that uses the ASCII Code conversion of Q-gram strings and comparing the BKVs according to these conversions instead of comparing strings directly. This idea can eliminate duplicated records to its minimum by finding them in a more accurate manner. However, experiments only covered English letters through their ASCII code. In a future work, we can test the results of using different languages and suggest how to work with Arabic letters which are the hardest language due its extra notations.

References

1. Issa, H.: Application of Duplicate Records Detection Techniques to Duplicate Payments in a Real Business Environment. Rutgers University, Rutgers Business School (2010)
2. Naderi, H.; Salehpour, N.; Farokhi, M.N.; Chegeni, B.H.: The search of new issues in the detection of near-duplicated documents. Int. J. Curr. Rev. **2**(2), 25–34 (2014)
3. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans. Knowl. Data Eng. **24**, 1537–1555 (2012)
4. Fellegi, I.P.; Sunter, A.B.: A theory for record linkage. J. Am. Stat. Soc. **64**(328), 1183–1210 (1969)
5. Hernandez, M.A.; Stolfo, S.J.: The merge/purge problem for large databases. In: Proceedings of the ACM SIGMOD'95, San Jose (1995)
6. Aizawa, A.; Oyama, K.: A fast linkage detection scheme for multi-source information integration. In: Proceedings of the IEEE International Workshop on Challenges in Web Information Retrieval and Integration WIRI'05, Tokyo, Japan (2005)
7. Cohen, W.W.; Richman, J.: Learning to Match and cluster large high-dimensional data sets for data integration. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACMSIGKDD'02, Edmonton, pp. 475–480 (2002)
8. Gravano, L.; Ipeirotis, P.G.; Jagadish, H.V.; Koudas, N.; Muthukrishnan, S.; Srivastava. D.: Approximate string joins in a database (Almost) for free. VLDB (2001)
9. Adrian, B.; Christian, B.; Sean, R.; Rainer, S.: High quality linkage using multibit trees for privacy-preserving blocking. Int. J. Popul. Data Sci. (IJPDS) **1**(1), 130 (2016)
10. Kevin, Z.; Peter, A.: A Q-gram birthmarking approach to predicting reusable hardware. In: Design, automation & test in Europe conference and exhibition (DATE), 14–18 March (2016)
11. Jie, L.; Haiying, Z.: Research and implementation of finding duplicate science project based on dimension filtering of Q-gram index. Destech Transactions on Engineering and Technology Research (2016)
12. Christen, P.: FEBRL: An open source data cleaning, deduplication and record linkage system with a graphical user interface. In: Pro-

ceeding of the 14th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'08), Las Vegas, USA, pp. 1065–1068, Aug. 24–27 (2008)
13. Elmagarmid, A.K.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
14. Alnoory, M.K.: Performance evaluation of similarity functions for duplicate record detection. M.Sc. Thesis, Yarmouk University (2011)
15. Churches, T.; Christen, P.; Lim, K.; Zhu, J.X.: Preparation of name and address data for record linkage using hidden Markov models. BioMed Cent. Med. Inf. Decis. Mak. **2**(1), 9 (2002)
16. Rahm, E.; Do, H.H.: Data cleaning: problems and current approaches. IEEE Data Eng. Bull. **23**(4), 3–13 (2000)
17. Bilenko, M.; Mooney, R.J.: On evaluation and training set construction for duplicate detection. In: Proceedings of the ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage and Object Consolidation, Washington, DC, pp. 7–12 (2003)
18. Higazy, A.A.; Sarhan, A.M.; El Tobely, T.: Web-based Arabic/English duplicate record detection with nested blocking technique. In: Proceedings of the IEEE 8th International Conference on Computer Engineering and Systems (ICCES), Egypt, pp. 313–318 (2013)
19. Azman, S.: Efficient identity matching using static pruning Q-gram indexing approach. Decis. Support Syst. **73**, 97–108 (2015)
20. Ramadan, B.; Christen, P.: Unsupervised blocking key selection for real-time entity resolution. In: Advances in Knowledge Discovery and Data Mining Volume 9078 of the Series, Lecture Notes in Computer Science. Springer, pp. 574–585 (2015)
21. Kreft, S.; Navarro, G.: On compressing and indexing repetitive sequences. Theor. Comput. Sci. J. **483**, 115–133 (2013)
22. McCallum, A.; Nigam, K.; Ungar, L.H.: Efficient clustering of high-dimensional datasets with application to reference matching. In: Proceedings of the ACM International Conference Knowledge Discovery and Data Mining, ACM SIGKDD'00, Boston, pp. 169–178 (2000)
23. Christen, P.: A comparison of personal name matching: techniques and practical issues. In: Proceedings of the IEEE Workshop on Mining Complex Data, IEEE ICDM'06, Hong Kong (2006)
24. Kumar, A.; Ingle, Y.S.; Pande, A.; Dhule, P.: Canopy clustering: a review on pre-clustering approach to K-means clustering. Int. J. Innov. Adv. Comput. Sci. (IJIACS) **3**(5), 22–29 (2014)
25. Cohen, W.W.; Ravikumar, P.; Fienberg, S.: A comparison of string distance metrics for name-matching tasks. In: Proceedings of the Workshop on Information Integration on the Web, held at IJCAI'03, Acapulco (2003)
26. Christen, P.; Goiser, K.: Quality and complexity measures for data linkage and deduplication. In: Guillet, F., Hamilton, H. (eds.) Quality Measures in Data Mining Series. Studies in Computational Intelligence, pp. 127–151. Springer, Berlin (2007)
27. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**(1), 31–88 (2001)
28. Shannon, C.E.: A mathematical theory of communications. Bell Syst. Technol. **27**, 379–423 (1948)
29. Ukkonen, E.: Approximate string matching with q-grams and maximal matches. Theory Comput. Sci. **92**, 191–211 (1992)
30. Kukich, K.: Spelling correction for the telecommunications network for the deaf. Commun. ACM **35**, 80–90 (1992)
31. Gravano, L.; Ipeirotis, P.G.; Koudas, N.; Srivastava, D.: Text joins for data cleansing and integration in an RDBMS. In: Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE) (2003)
32. Naumann, F.; Herschel, M.: An Introduction to Duplicate Detection. Morgan and Claypool Publishers, San Rafael (2010)
33. Christen, P.; Goiser, K.: A comparison of personal name matching: techniques and practical issues. In: Proceeding of Data Mining Workshops, ICDM Workshops (2006)