A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication

Peter Christen

Abstract—Record linkage is the process of matching records from several databases that refer to the same entities. When applied on a single database, this process is known as deduplication. Increasingly, matched data are becoming important in many application areas, because they can contain information that is not available otherwise, or that is too costly to acquire. Removing duplicate records in a single database is a crucial step in the data cleaning process, because duplicates can severely influence the outcomes of any subsequent data processing or data mining. With the increasing size of today's databases, the complexity of the matching process becomes one of the major challenges for record linkage and deduplication. In recent years, various indexing techniques have been developed for record linkage and deduplication. They are aimed at reducing the number of record pairs to be compared in the matching process by removing obvious nonmatching pairs, while at the same time maintaining high matching quality. This paper presents a survey of 12 variations of 6 indexing techniques. Their complexity is analyzed, and their performance and scalability is evaluated within an experimental framework using both synthetic and real data sets. No such detailed survey has so far been published.

Index Terms—Data linkage, data matching, entity resolution, index techniques, blocking, experimental evaluation, scalability.

1 Introduction

As many businesses, government agencies, and research projects collect increasingly large amounts of data, techniques that allow efficient processing, analyzing, and mining of such massive databases have in recent years attracted interest from both academia and industry. One task that has been recognized to be of increasing importance in many application domains is the matching of records that relate to the same entities from several databases. Often, information from multiple sources needs to be integrated and combined in order to improve data quality, or to enrich data to facilitate more detailed data analysis. The records to be matched frequently correspond to entities that refer to people, such as clients or customers, patients, employees, tax payers, students, or travelers.

The task of record linkage is now commonly used for improving data quality and integrity, to allow reuse of existing data sources for new studies, and to reduce costs and efforts in data acquisition [1]. In the health sector, for example, matched data can contain information that is required to improve health policies, information that traditionally has been collected with time consuming and expensive survey methods [2], [3]. Linked data can also help in health surveillance systems to enrich data that are used for the detection of suspicious patterns, such as outbreaks of contagious diseases.

Statistical agencies have employed record linkage for several decades on a routinely basis to link census data for further analysis [4]. Many businesses use deduplication and record linkage techniques with the aim to deduplicate their databases to improve data quality or compile mailing lists, or to match their data across organizations, for example, for collaborative marketing or e-Commerce projects. Many government organizations are now increasingly employing record linkage, for example within and between taxation offices and departments of social security to identify people who register for assistance multiple times, or who work and collect unemployment benefits.

Other domains where record linkage is of high interest are fraud and crime detection, as well as national security [5]. Security agencies and crime investigators increasingly rely on the ability to quickly access files for a particular individual under investigation, or cross-check records from disparate databases, which may help to prevent crimes and terror by early intervention.

The problem of finding records that relate to the same entities not only applies to databases that contain information about people. Other types of entities that sometimes need to be matched include records about businesses, consumer products, publications, and bibliographic citations, web pages, web search results, or genome sequences. In bioinformatics, for example, record linkage techniques can help find genome sequences in large data collections that are similar to a new, unknown sequence. In the field of information retrieval, it is important to remove duplicate documents (such as web pages and bibliographic citations) in the results returned by search engines, in digital libraries or in automatic text indexing systems [6], [7]. Another application of growing interest is finding and comparing consumer products from different online stores. Because product descriptions are often slightly varying, matching them becomes challenging [8].

In situations where unique entity identifiers (or keys) are available across all the databases to be linked, the problem of

• *The author is with the Research School of Computer Science, College of Engineering and Computer Science, The Australian National University, CSIT Building 108, Canberra ACT 0200, Australia. E-mail: peter.christen@anu.edu.au.*

Manuscript received 28 Sept. 2010; revised 26 Mar. 2011; accepted 14 May 2011; published online 7 June 2011.
Recommended for acceptance by C. Clifton.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2010-09-0524.
Digital Object Identifier no. 10.1109/TKDE.2011.127.

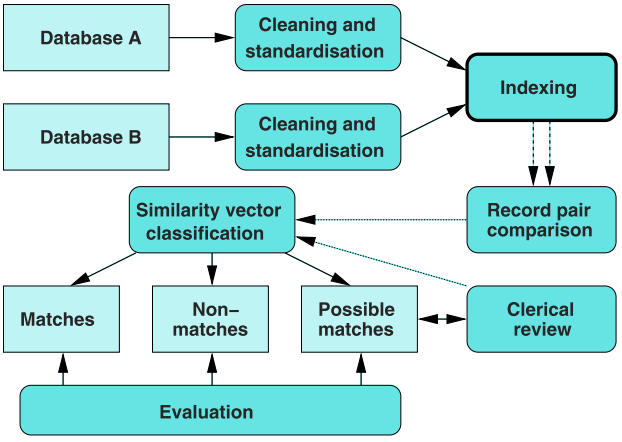

Fig. 1. Outline of the general record linkage process. The indexing step (the topic of this survey) generates candidate record pairs, while the output of the comparison step is vectors containing numerical similarity values.

matching records at the entity level becomes trivial: a simple database join is all that is required. However, in most cases no such unique identifiers are shared by all databases, and more sophisticated linkage techniques are required. These techniques can be broadly classified into deterministic, probabilistic, and learning-based approaches [4], [9], [10].

While statisticians and health researchers commonly name the task of matching records as data or record linkage [11] (the term used in this paper), the computer science and database communities refer to the same process as data or field matching [12], data integration [13], data scrubbing or cleaning [14], [15], data cleansing [16], duplicate detection [17], [18], information integration [19], entity resolution [20], [21], [22], reference reconciliation [23], or as the merge/purge problem [24]. In commercial processing of business mailing lists and customer databases, record linkage is usually seen as a component of *ETL* (extraction, transformation, and loading) tools. Two recent surveys have provided overviews of record linkage and deduplication techniques and challenges [4], [18].

1.1 The Record Linkage Process

Fig. 1 outlines the general steps involved in the linking of two databases. Because most real-world data are dirty and contain noisy, incomplete, and incorrectly formatted information, a crucial first step in any record linkage or deduplication project is data cleaning and standardization [25]. It has been recognized that a lack of good quality data can be one of the biggest obstacles to successful record linkage [2]. The main task of data cleaning and standardization is the conversion of the raw input data into well defined, consistent forms, as well as the resolution of inconsistencies in the way information is represented and encoded [15], [25].

The second step ("Indexing") is the topic of this survey, and will be discussed in more detail in Section 2. The indexing step generates pairs of candidate records that are compared in detail in the comparison step using a variety of comparison functions appropriate to the content of the record fields (attributes). Approximate string comparisons, which take (typographical) variations into account, are commonly used on fields that, for example, contain name and address details [12], while comparison functions specific for date, age, and numerical values are used for fields that contain such data [26]. Several fields are normally compared for each record pair, resulting in a vector that contains the numerical similarity values calculated for that pair.

Using these similarity values, the next step in the record linkage process is to classify the compared candidate record pairs into matches, nonmatches, and possible matches, depending upon the decision model used [9], [27]. Record pairs that were removed in the indexing step are classified as nonmatches without being compared explicitly. The majority of recent research into record linkage has concentrated on improving the classification step, and various classification techniques have been developed. Many of them are based on machine learning approaches [10], [20], [28], [29], [30], [31]. If record pairs are classified into possible matches, a clerical review process is required where these pairs are manually assessed and classified into matches or nonmatches. This is usually a time-consuming, cumbersome, and error-prone process, especially when large databases are being linked or deduplicated. Measuring and evaluating the quality and complexity of a record linkage project is a final step in the record linkage process [9].

1.2 Contributions

While various indexing techniques for record linkage and deduplication have been developed in recent years, so far no thorough theoretical or experimental survey of such techniques has been published. Earlier surveys have compared four or less indexing techniques only [32], [33]. It is therefore currently not clear which indexing technique is suitable for what type of data and what kind of record linkage or deduplication application. The aim of this survey is to fill this gap, and provide both researchers and practitioners with information about the characteristics of a variety of indexing techniques, including their scalability to large data sets, and their performance for data with different characteristics.

The contributions of this paper are a detailed discussion of six indexing techniques (with a total of 12 variations of them), a theoretical analysis of their complexity, and an empirical evaluation of these techniques within a common framework on a variety of both real and synthetic data sets.

The reminder of this paper is structured as follows: in the following Section 2 the indexing step of the record linkage process is discussed in more detail. The six indexing techniques are then presented in Section 3, followed by their experimental evaluation in Section 4. The results of these experiments are discussed in Section 5. An overview of related work is then provided in Section 6, and the paper is concluded in Section 7 with an outlook to future challenges and work in this area.

2 Indexing for Record Linkage and Deduplication

When two databases, $\mathbf{A}$ and $\mathbf{B}$, are to be matched, potentially each record from $\mathbf{A}$ needs to be compared with every record from $\mathbf{B}$, resulting in a maximum number of $|\mathbf{A}| \times |\mathbf{B}|$ comparisons between two records (with $|\cdot|$ denoting the number of records in a database). Similarly, when deduplicating a single database $\mathbf{A}$, the maximum number of possible

TABLE 1
Example Records and Blocking Keys

Record fields					Blocking keys and BKVs		
Identifiers	Givennames	Surnames	Postcodes	Suburb names	Sndx(GiN)+PC	Fi2D(PC)+DMe(SurN)	Sndx(SubN)+La2D(PC)
R1	Peter	Christen	2010	North Sydney	P360-2010	**20-KRST**	N632-10
R2	Pedro	Kristen	2000	Sydeny	P360-2000	**20-KRST**	S530-00
R3	Paul	Smith	2600	Canberra	P400-2600	26-SM0	**C516-00**
R4	Pablo	Smyth	2700	Canberra Sth	P140-2700	27-SM0	**C516-00**

How the blocking key values are generated is detailed in Section 2. The two highlighted bold pairs of BKVs illustrate that these records would be inserted into the same blocks.

comparisons is $|\mathbf{A}| \times (|\mathbf{A}| - 1)/2$, because each record in $\mathbf{A}$ potentially needs to be compared with all other records.

The performance bottleneck in a record linkage or deduplication system is usually the expensive detailed comparison of field (attribute) values between records [9], [32], making the naïve approach of comparing all pairs of records not feasible when the databases are large. For example, the matching of two databases with 1 million records each would result in 10^{12} (one trillion) possible record pair comparisons.

At the same time, assuming there are no duplicate records in the databases to be matched (i.e., one record in $\mathbf{A}$ can only be a true match to one record in $\mathbf{B}$ and vice versa), then the maximum possible number of true matches will correspond to $min(|\mathbf{A}|, |\mathbf{B}|)$. Similarly, for a deduplication the number of unique entities (and thus true matches) in a database is always smaller than or equal to the number of records in it. Therefore, while the computational efforts of comparing records increase quadratically as databases are getting larger, the number of potential true matches only increases linearly in the size of the databases.

Given this discussion, it is clear that the vast majority of comparisons will be between records that are not matches. The aim of the indexing step is to reduce this large number of potential comparisons by removing as many record pairs as possible that correspond to nonmatches. The traditional record linkage approach [4], [11] has employed an indexing technique commonly called *blocking* [32], which splits the databases into nonoverlapping blocks, such that only records within each block are compared with each other. A blocking criterion, commonly called a *blocking key* (the term used in this paper), is either based on a single record field (attribute), or the concatenation of values from several fields.

Because real-world data are often dirty and contain variations and errors [34], an important criteria for a good blocking key is that it can group similar values into the same block. What constitutes a "similar" value depends upon the characteristics of the data to be matched. Similarity can refer to similar sounding or similar looking values based on phonetic or character shape characteristics. For strings that contain personal names, for example, phonetic similarity can be obtained by using phonetic encoding functions such as Soundex, NYSIIS, or Double-Metaphone [35]. These functions, which are often language or domain specific, are applied when the blocking key values (BKVs) are generated.

As an example, Table 1 shows three different blocking keys and the resulting BKVs for four records. The first one is made of Soundex (Sndx) encoded givenname (GiN) values concatenated with full postcode (PC) values, the second consists of the first two digits (Fi2D) of postcode values concatenated with Double-Metaphone (DMe) encoded surname (SurN) values, and the third is made of Soundex encoded suburb name (SubN) values concatenated with the last two digits (La2D) of postcode values. To illustrate the two components of each blocking key in Table 1, their values are separated by a hyphen ("-"), however in real-world applications they would be concatenated directly.

Several important issues need to be considered when record fields are selected to be used as blocking keys. The first issue is that the quality of the values in these fields will influence the quality of the generated candidate record pairs. Ideally, fields containing the fewest errors, variations, or missing values should be chosen. Any error in a field value used to generate a BKV will potentially result in records being inserted into the wrong block, thus leading to missing true matches [9]. One approach used to overcome errors and variations is to generate several blocking keys based on different record fields, as is illustrated in Table 1. The hope is that records that refer to true matches have at least one BKV in common, and will therefore be inserted into the same block.

A second issue that needs to be considered when defining blocking keys is that the frequency distribution of the values in the fields used for blocking keys will affect the size of the generated blocks. Often this will be the case even after phonetic or other encodings have been applied. For example, a field containing surnames in a database from the United Kingdom, US, or Australia will likely contain a large portion of records with the value "Smith," which will results in a similarly large portion of records with the corresponding Soundex encoding "S530." If m records in database $\mathbf{A}$ and n records in database $\mathbf{B}$ have the same BKV, then $m \times n$ candidate record pairs will be generated from the corresponding block. The largest blocks generated in the indexing step will dominate execution time of the comparison step, because they will contribute a large portion of the total number of candidate record pairs. Therefore, it is of advantage to use fields that contain uniformly distributed values because they will result in blocks of equal sizes.

When blocking keys are defined, there is also a tradeoff that needs to be considered. On one hand, having a large number of smaller blocks will result in fewer candidate record pairs that will be generated. This will likely increase the number of true matches that are missed. On the other hand, blocking keys that result in larger blocks will generate an increased number of candidate record pairs that likely will cover more true matches, at the cost of having to compare more candidate pairs [32]. As will be discussed in the following section, some indexing techniques do allow explicit control of the size of the blocks that will be generated,

Identifiers	Surnames	BKVs (Soundex encoding)
R1	Smith	S530
R2	Miller	M460
R3	Peters	P362
R4	Myler	M460
R5	Smyth	S530
R6	Millar	M460
R7	Smyth	S530
R8	Miller	M460

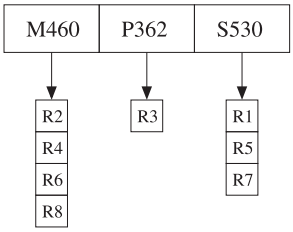

Fig. 2. Example records with surname values and their Soundex encodings used as BKVs, and the corresponding inverted index data structure as used for traditional blocking.

while for others the block sizes depend upon the characteristics of the record fields used in blocking keys.

All indexing techniques discussed in the following section do require some form of blocking key to be defined. The question of how to optimally choose record fields for blocking keys, such that as many true matching pairs as possible are included in the set of candidate record pairs, is orthogonal to the selection of an actual indexing technique. Traditionally, blocking keys have been manually selected by domain experts according to their understanding of the databases to be matched, or based on initial data exploration steps conducted.

In order to achieve an indexing that generates candidate record pairs of good quality, many recently developed indexing techniques require various parameters to be set. The optimal values of these parameters depend both upon the data to be matched (such as distribution of values and error characteristics), as well as the choice of blocking key(s) used. This makes it often difficult in practice to achieve a good indexing, because time consuming manual parameter tuning, followed by test linkages and careful evaluation is required. Additionally, in many real-world record linkage or deduplication applications, no data that contain the known true match status of record pairs are available that can be used to assess linkage quality [9]. Therefore, it is often not known how many true matches are included in the set of candidate record pairs. Measures suitable for assessing record linkage quality and complexity will be discussed in Section 4.2. Ideally, an indexing technique for record linkage and deduplication should be robust with regard to the selected parameter values or not require parameters at all, which would allow automated indexing [36].

3 Indexing Techniques

In this section, the traditional blocking approach and five more recently developed indexing techniques and variations of them are discussed in more detail. Their complexity is analyzed as the estimated number of candidate record pairs that will be generated. Knowing this number, together with a measured average time per record pair comparison (shown in Table 4), will allow an estimate of the runtime of the comparison step. Given this step is often the most time consuming step in a record linkage or deduplication project, such estimates will help users to predict how long a certain linkage or deduplication project will take.

The estimated number of candidate record pairs will be calculated for two different frequency distributions of BKVs. The first assumes a uniform distribution of values, resulting in each block containing the same number of records. The second assumes that the frequencies of the BKVs follow Zipf's law [37], a frequency distribution that is commonly found in data sets that contain values such as personal names [38]. Zipf's law states that in a list of words ranked according to their frequencies, the word at rank r has a relative frequency that corresponds to $1/r$. For attributes such as postcode, suburb name or age, the frequency distribution of their values is likely somewhere between a uniform and a Zipf-like frequency distribution. Therefore, assuming these two distributions should provide lower and upper limits of the number of candidate record pairs that can be expected when linking or deduplicating real-world databases.

Conceptually, the indexing step of the record linkage process can be split into the following two phases:

1. **Build.** All records in the database (or databases) are read, their BKVs are generated, and records are inserted into appropriate index data structures. For most indexing techniques, an inverted index [37] can be used. The BKVs will become the keys of the inverted index, and the record identifiers of all records that have the same BKV will be inserted into the same inverted index list. Fig. 2 illustrates this for a small example data set.

   When linking two databases, either a separate index data structure is built for each database, or a single data structure with common key values is generated. For the second case, each record identifier needs to include a flag that indicates from which database the record originates.

   The field values required in the comparison step need to be inserted into another data structure that allows efficient access to single random records when they are required for field comparisons. This can be achieved using an appropriately indexed database or hash table.
2. **Retrieve.** For each block, its list of record identifiers is retrieved from the inverted index, and candidate record pairs are generated from this list. For a record linkage, all records in a block from one database will be paired with all records from the block with the same BKV from the other database, while for a deduplication each record in a block will be paired with all other records in the same block. For example, from the block with key "S530" from Fig. 2 the pairs (R1, R5), (R1, R7), and (R5, R7) will be generated.

   The candidate record pairs are then compared in detail in the comparison step, and the resulting vectors containing numerical similarity values are given to a classifier in the classification step.

This survey mainly considers the **Build** phase, namely how different indexing techniques, using the same blocking key definition, are able to index records from data sets with different characteristics, and how this, in combination with various parameter settings, affects the number and quality of the candidate record pairs generated. The specific questions of interest are how many candidate pairs are generated, and how many of them are true matches and how many true nonmatches.

The following notation will be used when discussing the complexity of indexing techniques: $n_A = |\mathbf{A}|$ and $n_B = |\mathbf{B}|$ are the number of records in databases $\mathbf{A}$ and $\mathbf{B}$, respectively. For simplicity, it is assumed that only one blocking key definition is used, and that the BKVs generated for both databases are the same, i.e., if $\mathbf{K}_A$ and $\mathbf{K}_B$ are the sets of BKVs generated from $\mathbf{A}$ and $\mathbf{B}$, then $\mathbf{K}_A \equiv \mathbf{K}_B$. While this is a rare scenario in most real-world applications, it provides an upper bound on the number of candidate record pairs, because any BKV that is only generated by one of the two databases will not result in any candidate record pairs. The number of different BKVs is denoted as b, with $b = |\mathbf{K}_A| = |\mathbf{K}_B|$ (and $|\cdot|$ denoting the number of elements in a set).

3.1 Traditional Blocking

This technique has been used in record linkage since the 1960s [11]. All records that have the same BKV are inserted into the same block, and only records within the same block are then compared with each other. Each record is inserted into one block only (assuming a single blocking key definition). As illustrated in Fig. 2, traditional blocking can be implemented efficiently using a standard inverted index [37], as described in the **Build** phase above. In the **Retrieve** phase, the identifiers of all records in the same block are retrieved and the corresponding candidate record pairs are generated.

While traditional blocking does not have any explicit parameters, the way blocking keys are defined will influence the quality and number of candidate record pairs that are generated. As discussed in Section 2, a major drawback of traditional blocking is that errors and variations in the record fields used to generate BKVs will lead to records being inserted into the wrong block. This drawback can be overcome by using several blocking key definitions based on different record fields, or different encodings applied on the same record fields. A second drawback of traditional blocking is that the sizes of the blocks generated depend upon the frequency distribution of the BKVs, and thus it is difficult in practice to predict the total number of candidate record pairs that will be generated.

If a uniform distribution of field values is assumed that leads to uniformly distributed BKVs, then all blocks will be of uniform size and contain n_A/b or n_B/b records, respectively, with b being the number of different BKVs. In this situation, the number of candidate record pairs generated for a record linkage equals

$$u_{TBRL} = b \times \left(\frac{n_A}{b} \times \frac{n_B}{b}\right) = \frac{n_A n_B}{b}, \quad (1)$$

while for a deduplication the number of candidate record pairs generated equals

$$u_{TBD} = b \times \left(\frac{n_A}{b} \times \left(\frac{n_A}{b} - 1\right) \Big/ 2\right) = \frac{n_A}{2}\left(\frac{n_A}{b} - 1\right). \quad (2)$$

For both situations, this corresponds to a b-fold reduction in the number of candidate pairs compared to the naïve approach of comparing all records with each other.

If a Zipf frequency distribution of record field values is assumed that leads to Zipf-like distribution of BKVs, then the size of the generated blocks will also follow a Zipf-like frequency distribution. With b blocks, the number of candidate record pairs generated for a record linkage in this situation equals

$$\begin{aligned} z_{TBRL} &= \sum_{i=1}^{b}\left(\frac{1/i}{H_b} \times n_A\right) \times \left(\frac{1/i}{H_b} \times n_B\right) \\ &= \frac{n_A n_B}{H_b^2} \times \sum_{i=1}^{b} \frac{1}{i^2}, \end{aligned} \quad (3)$$

with H_b being the harmonic number of the partial harmonic sum, $H_b = \sum_{i=1}^{b} 1/i$. For a deduplication, the number of candidate record pairs generated equals

$$\begin{aligned} z_{TBD} &= \sum_{i=1}^{b}\left(\frac{1/i}{H_b} \times n_A\right) \times \left(\frac{1/i}{H_b} \times n_A - 1\right) \Big/ 2 \\ &= \frac{1}{2} \times \left(\frac{n_A^2}{H_b^2}\sum_{i=1}^{b}\frac{1}{i^2} - \frac{n_A}{H_b}\sum_{i=1}^{b}\frac{1}{i}\right). \end{aligned} \quad (4)$$

For a given number of blocks and a database (or databases) of a given size, having blocks of uniform size will lead to the smallest number of candidate records pairs generated compared to any nonuniform distribution. For example, for two equal sized blocks each containing x records, the number of candidate record pairs generated will equal $2 \times x^2$. Changing the distribution to blocks containing $(x+1)$ and $(x-1)$ records, respectively, will result in $(x+1)^2 + (x-1)^2 = (x^2 + 2x + 1) + (x^2 - 2x + 1) = 2 \times x^2 + 2$. Therefore, every redistribution away from uniform block sizes will increase the number of candidate record pairs generated. From this follows that $u_{TBRL} < z_{TBRL}$ and $u_{TBD} < z_{TBD}$ for the same number of blocks and same number of records in the database(s) to be linked or deduplicated.

3.2 Sorted Neighborhood Indexing

This technique was first proposed in the mid 1990s [24]. Its basic idea is to sort the database(s) according to the BKVs, and to sequentially move a window of a fixed number of records w ($w > 1$) over the sorted values. Candidate record pairs are then generated only from records within a current window. As illustrated in Figs. 3 and 4, there are two different approaches of how this technique can be implemented.

3.2.1 Sorted Array-Based Approach

In this first approach, as originally proposed [24], the BKVs are inserted into an array that is sorted alphabetically, as shown in the left-hand side of Fig. 3. The window is then moved over this sorted array and candidate record pairs are generated from all records in the current window, as illustrated in the right-hand side of Fig. 3. In case of a record linkage, the BKVs from both databases will be

Window positions	BKVs (Surname)	Identifiers
1	Millar	R6
2	Miller	R2
3	Miller	R8
4	Myler	R4
5	Peters	R3
6	Smith	R1
7	Smyth	R5
8	Smyth	R7

Window range	Candidate record pairs
1 – 3	(R6,R2), (R6,R8), (R2,R8)
2 – 4	(R2,R8), (R2,R4), (R8,R4)
3 – 5	(R8,R4), (R8,R3), (R4,R3)
4 – 6	(R4,R3), (R4,R1), (R3,R1)
5 – 7	(R3,R1), (R3,R5), (R1,R5)
6 – 8	(R1,R5), (R1,R7), (R5,R7)

Fig. 3. Example sorted neighborhood technique based on a sorted array, with BKVs being the surname values from Fig. 2 (and the corresponding record identifiers), and a window size $w = 3$.

inserted into one combined array and then sorted alphabetically, but candidate record pairs are generated in such a way that for each pair one record is selected from each of the two databases.

For a record linkage, assuming the length of the sorted array is $(n_A + n_B)$ (the total number of records in both databases), then the number of window positions equals $(n_A + n_B - w + 1)$, while for a deduplication the number of windows is $(n_A - w + 1)$. As can be seen in the right-hand side of Fig. 3, most candidate record pairs are generated in several windows, however, each unique pair will only be compared once in the comparison step. Because the window size is fixed in this approach, the number of candidate record pairs generated is independent of the frequency distribution of the BKVs, and only depends upon the window size w and the size(s) of the database(s). If $\alpha = n_A/(n_A + n_B)$ denotes the ratio of the number of records in database **A** over the number of records in both databases, and $\beta = n_B/(n_A + n_B) = (1 - \alpha)$ the corresponding ratio for database **B**, then for a record linkage the number of unique candidate record pairs generated equals

$$\begin{aligned} u_{SNRL_{SA}} &= (\alpha w)(\beta w) + (n_A + n_B - w) \\ &\quad \times (\alpha((w-1)\beta) + \beta((w-1)\alpha)) \\ &= \alpha\beta w^2 + 2\alpha\beta(n_A + n_B - w)(w-1) \\ &= \alpha\beta(w^2 + 2(n_A + n_B - w)(w-1)) \\ &= \frac{n_A n_B}{(n_A + n_B)^2} \\ &\quad \times (w^2 + 2(n_A + n_B - w)(w-1)). \end{aligned} \tag{5}$$

The first term in the first line equals to the number of candidate record pairs that are generated in the first window position, while the remainder of the equation equals to the number of unique pairs generated in the remaining $(n_A + n_B - w)$ window positions. Assuming evenly mixed BKVs from **A** and **B**, in the first window position there will be αw records from database **A** that have to be compared to βw records from database **B**. For all following window positions, with a likelihood of α the newest record added to the window originates from database **A**, and has to be compared with $(w-1)\beta$ records in the current window that are from database **B**. A similar calculation, the term $\beta((w-1)\alpha)$, can be made when the newest record in a window originates from database **B**. The total number of candidate record pairs generated depends quadratically upon the window size w, and on the harmonic mean of the sizes of the two databases that are linked.

For a deduplication, the number of unique candidate pairs generated (duplicate pairs not counted) equals

$$\begin{aligned} u_{SND_{SA}} &= w(w-1)/2 + (n_A - w)(w-1) \\ &= (w-1)\Big(n_A - \frac{w}{2}\Big). \end{aligned} \tag{6}$$

For both a record linkage and a deduplication the number of candidate record pairs generated is independent of the frequency distribution of the BKVs, and therefore no analysis of Zipf-distributed values is required.

A major drawback of this approach is that if a small window size is chosen, it might not be large enough to cover all records that have the same BKV. For example, there might be several thousand records with a surname value "Smith" in a large database, but with a window size of, for example, $w = 10$ not all of these records will be in the same current window, and thus not all of them will be compared with each other. One solution to this problem is to select blocking keys that are made of the concatenation of several record fields (like surname and given name), so that they have a large number of different values, rather than employing encoding functions that group many similar field values together.

Another problem with this approach is that the sorting of the BKVs is sensitive toward errors and variations in the first few positions of values. For example, if given names are used as BKVs, then "Christina" and "Kristina" will very likely be too far away in the sorted array to be inserted into the same window, even though they are very similar names and might refer to the same person. This drawback can be overcome by employing several blocking key definitions based on different record fields, or by defining blocking

Window positions	BKVs (Surname)	Identifiers
1	Millar	R6
2	Miller	R2, R8
3	Myler	R4
4	Peters	R3
5	Smith	R1
6	Smyth	R5, R7

Window range	Candidate record pairs
1 – 3	(R6,R2), (R6,R8), (R6,R4), (R2,R8), (R2,R4), (R8,R4)
2 – 4	(R2,R8), (R2,R4), (R2,R3), (R8,R4), (R8,R3), (R4,R3)
3 – 5	(R4,R3), (R4,R1), (R3,R1)
4 – 6	(R3,R1), (R3,R5), (R3,R7), (R1,R5), (R1,R7), (R5,R7)

Fig. 4. Example sorted neighborhood technique based on an inverted index and with the same BKVs and window size as in Fig. 3.

keys based on reversed field values (for example "anitsirhc" and "anitsirk").

The **Build** phase for this indexing approach also requires the sorting of the array, which has a complexity of $O(n\ log\ n)$, with $n = (n_A + n_B)$ for a record linkage, and $n = n_A$ for a deduplication.

3.2.2 Inverted Index-Based Approach

An alternative approach [39] for the sorted neighborhood technique is illustrated in Fig. 4. Rather than inserting BKVs into a sorted array, this approach utilizes an inverted index similar to traditional blocking. The index keys contain the alphabetically sorted BKVs, as is shown in the left-hand side of Fig. 4. The window is moved over these sorted BKVs, and candidate record pairs are formed from all records in the corresponding index lists, as illustrated in the right-hand side of Fig. 4. Similar to the sorted array-based approach, most candidate record pairs are generated in several windows, but each unique candidate pair will again only be compared once in the comparison step. The number of generated candidate record pairs with this approach depends upon the number of record identifiers that are stored in the inverted index lists.

For a window size $w = 1$, this inverted index-based approach reduces to traditional blocking as described in Section 3.1. For all window sizes $w > 1$, the generated candidate record pairs will therefore be a superset of the pairs generated by traditional blocking. In general, for two window sizes w_i and w_j, with $w_i < w_j$, all candidate record pairs generated with window size w_i will also be in the set of pairs generated with w_j. However, the larger the window size is, the larger the number of generated candidate record pairs becomes.

The number of window positions with this approach for both a record linkage and a deduplication is $(b - w + 1)$, with b being the number of different BKVs. For a record linkage, record identifiers from both databases will be inserted into a common inverted index data structure, together with a flag stating if a record originates from database **A** or **B**. Assuming a uniform distribution of BKVs, each inverted index list will contain $n_A/b + n_B/b$ record identifiers. The number of unique candidate record pairs generated for a record linkage is

$$\begin{aligned} u_{SNRL_{II}} &= w\frac{n_A}{b} \times w\frac{n_B}{b} + (b-w) \\ &\quad \times \left(\frac{n_A}{b} \times w\frac{n_B}{b} + \frac{n_B}{b} \times (w-1)\frac{n_A}{b}\right) \\ &= w^2\frac{n_A n_B}{b^2} + (b-w) \times (2w-1)\frac{n_A n_B}{b^2} \\ &= \frac{n_A n_B}{b^2}\big(w^2 + (b-w)(2w-1)\big). \end{aligned} \tag{7}$$

The first term in (7) corresponds to the number of candidate record pairs generated in the first window position, while the second term corresponds to the $(b - w)$ following window positions. The first part of this second term refers to the candidate pairs that are generated between the record identifiers in the most recently added inverted index list in the current window that come from database **A** and the identifiers in the previous lists from database **B**, while the second part refers to the pairs that are generated between the newest list from the index of database **B** and the previous lists from the index of database **A**. If the window size is set to $w = 1$, the above formula reduces to (1), generating the same number of candidate record pairs as with traditional blocking, because the term $(w^2 + (b - w)(2w - 1)) = (1^2 + (b - 1)(2 - 1)) = 1 + (b - 1) = b$.

For a deduplication, the number of unique candidate pairs generated equals

$$\begin{aligned} u_{SND_{II}} &= w\frac{n_A}{b} \times \left(w\frac{n_A}{b} - 1\right)\Big/2 + (b-w) \\ &\quad \times \left(\frac{n_A}{b} \times (w-1)\frac{n_A}{b} + \frac{n_A}{b}\left(\frac{n_A}{b} - 1\right)\Big/2\right) \\ &= w\frac{n_A}{2b} \times \left(w\frac{n_A}{b} - 1\right) + (b-w) \\ &\quad \times \left(\frac{n_A^2}{b^2}(w-1) + \frac{n_A}{2b}\left(\frac{n_A}{b} - 1\right)\right). \end{aligned} \tag{8}$$

As can easily be verified, with $w = 1$ the above formula reduces to (2).

For BKVs that have a Zipf-like frequency distribution, calculating the number of candidate record pairs is difficult, because this number will depend upon the ordering of the inverted index lists. In the worst case scenario, the size of the inverted index lists (in number of record identifiers they contain) corresponds to the alphabetically sorted BKVs, such that the longest list is the alphabetically first, the second longest the alphabetically second, and so on. As a result, the first window would contain the largest number of record identifiers. Following (3) and (7), the number of unique candidate record pairs generated in this worst case scenario for a record linkage equals

$$\begin{aligned} z_{SNRL_{II}} &= \left(\frac{n_A}{H_b}\sum_{i=1}^{w}\frac{1}{i}\right) \times \left(\frac{n_B}{H_b}\sum_{i=1}^{w}\frac{1}{i}\right) \\ &\quad + \sum_{j=2}^{b-w+1}\left(\frac{n_A}{H_b(j+w-1)}\left(\sum_{i=j}^{j+w-1}\frac{n_B}{H_b i}\right)\right. \\ &\quad \left. + \frac{n_B}{H_b(j+w-1)}\left(\sum_{i=j}^{j+w-2}\frac{n_A}{H_b i}\right)\right). \end{aligned} \tag{9}$$

Using (4) and (8), for a deduplication the number of candidate record pairs can be calculated as

$$\begin{aligned} z_{SND_{II}} &= \frac{1}{2}\left(\frac{n_A}{H_b}\sum_{i=1}^{w}\frac{1}{i}\right) \times \left(\frac{n_A}{H_b}\sum_{i=1}^{w}\frac{1}{i} - 1\right) \\ &\quad + \sum_{j=2}^{b-w+1}\left(\frac{n_A}{H_b(j+w-1)}\left(\sum_{i=j}^{j+w-2}\frac{n_A}{H_b i}\right)\right. \\ &\quad \left. + \frac{n_A}{2H_b(j+w-1)} \times \left(\frac{n_A}{H_b(j+w-1)} - 1\right)\right). \end{aligned}$$

The inverted index-based sorted neighborhood approach has two main disadvantages. First, similarly to traditional blocking, the largest blocks will dominate the number of candidate record pairs that are generated, and therefore also dominate the time requirements of the comparison step. The second disadvantage is that the sorting of the BKVs assumes that their beginning is error free. Otherwise,

Identifiers	BKVs (Surname)	Bigram sub-lists	Index key values
R1	Smith	[sm,mi,it,th], [mi,it,th], [sm,it,th], [sm,mi,th], [sm,mi,it]	**smmiitth**, miitth, smitth, smmith, smmiit
R2	Smithy	[sm,mi,it,th,hy], [mi,it,th,hy], [sm,it,th,hy], [sm,mi,th,hy], [sm,mi,it,hy], [sm,mi,it,th]	smmiitthhy, miitthhy, smitthhy, smmithhy, smmiithy, **smmiitth**
R3	Smithe	[sm,mi,it,th,he], [mi,it,th,he], [sm,it,th,he], [sm,mi,th,he], [sm,mi,it,he], [sm,mi,it,th]	smmiitthhe, miitthhe, smitthhe, smmithhe, smmiithe, **smmiitth**

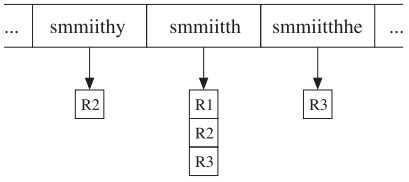

Fig. 5. Q-gram-based indexing with surnames used as BKVs, index key values based on bigrams ($q = 2$), and calculated using a threshold set to $t = 0.8$. The right-hand side shows three of the resulting inverted index lists (blocks), with the common BKV highlighted in bold in the index key value column.

similar values will not be close enough in the sorted keys of the inverted index and might therefore not be covered in the same window.

As is recommended with the sorted array-based approach, it is therefore good practice to define several blocking keys, ideally based on different record fields, and run this indexing approach using each of these blocking keys. Similar to traditional blocking, for this sorted neighborhood approach it will also be of advantage to use preprocessing, like phonetic encodings [35], to group similar record values into the same blocks, i.e., convert them into the same BKVs.

Assuming that the number of BKVs is much smaller than the number of records in the database(s) to be matched or deduplicated (i.e., $b \ll (n_A + n_B)$ or $b \ll n_A$, respectively), the sorting of the BKVs will be much faster than for the sorted array-based approach because the sorting step has a complexity of $O(b\ log\ b)$.

3.2.3 Adaptive Sorted Neighborhood Approach

Recent research has looked at how the sorted neighborhood indexing technique based on a sorted array can be improved [40], [41]. The issue of having a fixed block size w which can result in missed true matches (because not all same BKVs fit into one window, as discussed in Section 3.2.1) has been addressed through an adaptive approach to dynamically set the window size [40]. Depending upon the characteristics of the BKVs used in the sorted array, the idea is to find values adjacent to each other that are significantly different from each other using an appropriate string similarity measure [35]. These so called *boundary pairs* of BKVs are then used to form blocks, i.e., they mark the positions in the sorted array where one window ends and a new one starts. This approach can therefore be seen as a combination of traditional blocking and the sorted neighborhood approach. Due to the adaptive nature of the approach, where block sizes are determined by the similarities between BKVs, a theoretical analysis of the number of generated candidate record pairs would depend upon the actual BKVs and therefore not be of general use. This adaptive approach will however be evaluated experimentally in Section 4.

Another recently developed approach generalizes traditional blocking and the sorted neighborhood technique, and combines them into a sorted blocks method [41]. The authors of this approach show that traditional blocking and sorted neighborhood are two ends of a general approach, with blocking corresponding to sorted neighborhood where the window is moved forward w positions rather than 1, resulting in nonoverlapping blocks. The proposed combined approach allows specification of the desired overlap, and experimental results presented show that the sorted neighborhood approach performs better than traditional blocking, especially for small blocks [41].

3.3 Q-Gram-Based Indexing

The aim of this technique is to index the database(s) such that records that have a similar, not just the same, BKV will be inserted into the same block. The basic idea is to create variations for each BKV using q-grams (substrings of lengths q), and to insert record identifiers into more than one block.

Each BKV is converted into a list of q-grams, and sublist combinations of these q-gram lists are then generated down to a certain minimum length, which is determined by a user-selected threshold t ($t \leq 1$). For a BKV that contains k q-grams, all sublist combinations down to a minimum length of $l = max(1, \lfloor k \times t \rfloor)$ will be created ($\lfloor \cdots \rfloor$ denotes rounding to the next lower integer number). These sublists are then converted back into strings and used as the actual key values into an inverted index, as is illustrated in Fig. 5.

Different from the inverted index used in traditional blocking is that each record identifier is generally inserted into several index lists, according to the number of q-gram sublists generated for its BKV. With a threshold $t = 1.0$ however, each record identifier will be inserted into one inverted index list only, and in this case q-gram-based indexing will generate the same candidate record pairs as traditional blocking.

Fig. 5 illustrates q-gram-based indexing for three example records, $q = 2$ (bigrams), and a threshold $t = 0.8$. The BKV "Smith" in the first record (R1), for example, contains four ($k = 4$) bigrams: "sm," "mi," "it," "th" (assuming all letters have been converted into lower case beforehand), and so the length l of the shortest sublists for this value can be calculated as $l = \lfloor 4 \times 0.8 \rfloor = 3$. Therefore, four sublists each containing three bigrams will be generated for this BKV: [mi, it, th], [sm, it, th], [sm, mi, th], and [sm, mi, it]. Each of these is generated by removing one of the four original bigrams. These sublists will then be converted back into strings to form the actual key values used in the inverted index, as is shown in Fig. 5. The identifier of the record R1 will therefore be inserted into the five inverted index lists with key values "smmiitth," "miitth," "smitth," "smmith," and "smmiit." With an even lower threshold ($t < 0.75$), sublists of length two would be generated recursively from the sublists of length three.

TABLE 2
Number of Bigram ($q = 2$) Sublists (s) Generated According to (10) for Different Threshold Values t and Different Number of q-Grams k in the BKVs

Number of bigrams in BKVs, k	Threshold value t			
	0.9	0.8	0.7	0.6
3	4	4	4	7
4	5	5	11	11
6	7	22	22	42
8	9	37	93	163
10	11	56	176	386
12	79	299	794	1586
15	121	576	4944	9949
20	211	6196	60,460	263,950

The number of sublists generated for a BKV depends both upon the number of q-grams it consists of, as well as the value of the threshold t. Lower values of t will lead to an increased number of shorter sublists, and therefore a larger number of different index key values. The longer a BKV is, the more sublists will be generated. For a BKV of length c characters, there will be $k = (c - q + 1)$ q-grams, and with $l = max(1, \lfloor k \times t \rfloor)$ the length of the shortest sublists, a total of

$$s = \sum_{i=l}^{k} \binom{k}{i}, \tag{10}$$

sublists will be generated from this BKV. From Table 2, it can be seen that the value of s grows exponentially with longer BKVs, and as the threshold t is set to lower values. The time required to generate the q-gram sublists will therefore be dominated by the recursive generation of sublists for longer BKVs.

Fig. 6 shows the length frequency distributions of values from three record fields that are common in many databases that contain information about people. As can be seen, these distributions roughly follow Poisson distributions with parameter $5 \leq \lambda \leq 10$. Therefore, assuming BKV lengths that follow a Poisson distribution, it is possible to estimate the overhead of q-gram-based indexing compared to traditional blocking.

Let v denote the average number of times each record identifier is inserted into an inverted index list (block), compared to being inserted into just one index list as is done with traditional blocking (i.e., $v = 1$). Assuming that the average length of BKVs in q-grams is λ, their maximum length is l_{max}, and the minimum sublist length threshold is t, then v can be calculated as

$$v = \sum_{l=1}^{l_{max}} \left(\frac{\lambda^l e^{-\lambda}}{l!} \sum_{i=max(1,\lfloor l \times t \rfloor)}^{l} \binom{l}{i} \right). \tag{11}$$

This summation covers BKVs of lengths 1 to l_{max}, and for each of them the number of sublists generated for this specific length is calculated by combining a Poisson distribution with (10).

Assuming there are b different BKVs, the question now is how many index key values (denoted with b') are generated by the q-gram sublist generation process. This number depends upon the characteristics of the data, specifically

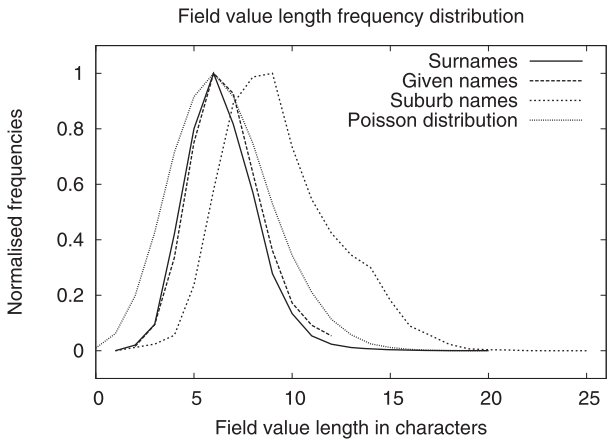

Fig. 6. Normalized length frequency distributions of record field values from an Australian telephone directory containing around seven million records. For comparison, a Poisson distribution (with $\lambda = 6.5$) is also shown.

the distribution of unique q-grams in the BKVs. One extreme situation would be that every BKV generates a set of unique index key values that are not shared with any other BKV, and thus $b' = v \times b$. The other extreme situation would be where every index key value is equal to an existing BKV, and thus $b' = b$.

Assuming uniform frequency distribution of the BKVs, (1) can be used to estimate the number of candidate record pairs that will be generated for a record linkage. With the two extreme situations described above, this number will be

$$vb \times \left(\frac{n_A v}{vb} \times \frac{n_B v}{vb} \right) \leq u_{QGRL} \leq b \times \left(\frac{n_A v}{b} \times \frac{n_B v}{b} \right),$$

which can be simplified to

$$\frac{n_A n_B v}{b} \leq u_{QGRL} \leq \frac{n_A n_B v^2}{b}. \tag{12}$$

Similar, for a deduplication, (2) can be used to estimate the number of candidate record pairs as

$$\frac{n_A v}{2} \left(\frac{n_A}{b} - 1 \right) \leq u_{QGD} \leq \frac{n_A v}{2} \left(\frac{n_A v}{b} - 1 \right). \tag{13}$$

For BKVs that follow a Zipf frequency distribution, (3) and (4) can be extended similarly to calculate the estimated number of candidate record pairs.

As was shown in an earlier study [32], q-gram-based indexing can lead to candidate record pairs that cover more true matches than both traditional blocking and the sorted neighborhood indexing techniques. The drawback, however, is that a much larger number of candidate record pairs will be generated, leading to a more time consuming comparison step. This will be confirmed in the experiments in Sections 4 and 5.

A similar q-grams-based approach to indexing has been proposed within a database framework [42], using q-gram-based similarity joins and filtering techniques to improve performance. This approach was implemented completely within a relational database and using SQL statements by generating auxiliary database tables that contain the q-grams and their record identifiers.

Identifiers	BKVs (Givenname)	Suffixes
R1	Catherine	catherine, atherine, therine, herine, erine, rine
R2	Katherina	katherina, atherina, therina, herina, erina, rina
R3	Catherina	catherina, atherina, therina, herina, erina, rina
R4	Catrina	catrina, atrina, trina, rina
R5	Katrina	katrina, atrina, trina, rina

Suffix	Identifiers
atherina	R2,R3
atherine	R1
atrina	R4,R5
catherina	R3
catherine	R1
catrina	R4
erina	R2,R3
erine	R1
herina	R2,R3

Suffix	Identifiers
herine	R1
katherina	R2
katrina	R5
~~rina~~	~~R2,R3,R4,R5~~
rine	R1
therina	R2,R3
therine	R1
trina	R4,R5

Fig. 7. Suffix-array-based indexing with given names used as BKVs, a minimum suffix length $l_m = 4$, and a maximum block size $b_M = 3$. The two tables on the right-hand side show the resulting sorted suffix array. The block with suffix value "rina" will be removed because it contains more than b_M record identifiers.

3.4 Suffix Array-Based Indexing

This technique has recently been proposed as an efficient domain independent approach to multisource information integration [19]. The basic idea is to insert the BKVs and their suffixes into a suffix array-based inverted index. A suffix array contains strings or sequences and their suffixes in an alphabetically sorted order. Indexing based on suffix arrays has successfully been used on both English and Japanese bibliographic databases [19].

In this indexing technique, only suffixes down to a minimum length, l_m, are inserted into the suffix array. For example, for a BKV "christen" and $l_m = 5$, the values "christen," "hristen," "risten" and "isten" will be generated, and the identifiers of all records that have this BKV will be inserted into the corresponding four inverted index lists. Fig. 7 shows several other examples of this approach. A BKV of length c characters will result in $(c - l_m + 1)$ suffixes to be generated. Similar to q-gram-based indexing, the identifier of a record will likely be inserted into several inverted index lists.

To limit the maximum size of blocks (and thus the number of candidate records pairs to be generated), a second parameter, b_M, allows the maximum number of record identifiers in a block to be set. Blocks that contain more that b_M record identifiers will be removed from the suffix array. For example, in Fig. 7, with $b_M = 3$ the block with suffix value "rina" will be removed because it contains four record identifiers.

To calculate the number of candidate record pairs that will be generated with suffix array-based indexing, similar to q-gram-based indexing the lengths of the BKVs needs to be estimated. Assuming a Poisson distribution of the length of BKVs and following (11), the average number of suffixes generated from a BKV can be calculated as

$$v = \sum_{l=l_m}^{l_M} \left(\frac{\lambda^l e^{-\lambda}}{l!} (l - l_m + 1) \right), \tag{14}$$

with l_M being the maximum and λ the average length (both in characters) of all BKVs in the database(s) to be matched or deduplicated.

Assuming there are b unique BKVs, the minimum number of suffix values generated would be b in the extreme situation where all BKVs are of length l_m characters, and thus no shorter suffixes are generated. The other extreme situation would occur when each BKV generates suffixes that are unique. In this situation, assuming each BKV in average generates v suffixes, a total of $v \times b$ unique suffix values will be generated. With the maximum size of each block being b_M, the number of candidate record pairs generated can be estimated as

$$b \times b_M^2 \leq u_{SARL} \leq bv \times b_M^2, \tag{15}$$

for a record linkage, and for a deduplication as

$$b \times \frac{b_M(b_M - 1)}{2} \leq u_{SAD} \leq b \times \frac{vb_M(b_M - 1)}{2}. \tag{16}$$

These estimates assume each block contains exactly b_M record identifiers. In practice, it is very unlikely this will occur, and thus less record pairs will be generated.

As can be seen in Fig. 7, one problem with suffix array-based indexing is that errors and variations at the end of BKVs will result in records being inserted into different blocks, potentially missing true matches. To overcome this drawback, a modification of the suffix generation process is to not only generate the true suffixes of BKVs, but all substrings down to the minimum lengths of l_m in a sliding window fashion. For example, for the BKV "christen" and $l_m = 5$, this approach would generate the substrings: "christen," "christe," "hristen," "christ," "hriste," "risten," "chris," "hrist," "riste," and "isten." This approach is similar to q-gram-based indexing as described in Section 3.3. It can better overcome errors and variations at different positions in the BKVs, at the costs of creating more blocks and inserting record identifiers into a larger number of blocks compared to the original suffix array technique. This proposed variation will be evaluated experimentally in Section 4.

3.4.1 Robust Suffix Array-Based Indexing

An improvement upon the original suffix array-based indexing technique has recently been proposed [43]. The idea is similar to adaptive blocking [40], in that the inverted index lists of suffix values that are similar to each other in the sorted suffix array are merged. An approximate string similarity measure [35] is calculated for all pairs of neighboring suffix values, and if the similarity of a pair is above a selected threshold t, then their lists are merged to form a new larger block.

For example, using the given name suffix values from Fig. 7, the normalized edit-distance string measure [35], and a minimum similarity of $t = 0.85$, then the following suffix string pairs and their corresponding record identifier lists will be merged into one block each: "atherina" and "atherine" (with similarity 0.875 and resulting in list R1, R2, R3), "catherina" and "catherine" (with similarity 0.889 and resulting in list R1, R3), and "therina" and "therine"

Identifiers	BKVs (Surname)	Sorted bigram lists
R1	Hanlan	[(an,2), (ha,1), (la,1), (nl,1)]
R2	Gansan	[(an,2), (ga,1), (ns,1), (sa,1)]
R3	Gargan	[(an,1), (ar,1), (ga,2), (rg,1),]

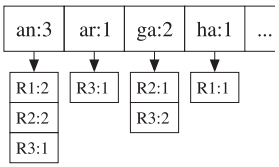

Fig. 8. Canopy clustering example with BKVs based on surnames, and their sorted bigram ($q = 2$) lists including DF counts. The TF and DF counts in the inverted index data structure are used to calculate TF-IDF weights.

(with similarity 0.857 and resulting in list R1, R2, R3). As will be shown in the experimental evaluation in Section 4, this indexing technique can lead to improved matching or deduplication results at the cost of larger blocks, and thus more candidate record pairs that need to be compared. A detailed analysis of the efficiency and time complexity of this approach has been presented elsewhere [43].

3.5 Canopy Clustering

This indexing technique is based on the idea of using a computationally cheap clustering approach to create high-dimensional overlapping clusters, from which blocks of candidate record pairs can then be generated [29], [44]. Clusters are created by calculating the similarities between BKVs using measures such as Jaccard or TF-IDF/cosine [37]. Both of these measures are based on tokens [29], [35], which can be characters, q-grams or words. They can be implemented efficiently using an inverted index which has tokens, rather than the actual BKVs, as index keys.

This inverted index data structure is built by converting BKVs into lists of tokens (usually q-grams), with each unique token becoming a key in the inverted index. All records that contain this token in their BKV will be added to the corresponding inverted index list. If the TF-IDF/cosine similarity is used, additional information has to be calculated and stored in the index. First, for each unique token, the number of records that contain this token is required. This corresponds to the term frequency (TF) of the token, and equals the number of record identifiers stored in a token's inverted index list. Second, within the inverted index lists themselves, the document frequency (DF) of a token (i.e., how often it appears in a BKV) needs to be stored. Fig. 8 shows an example of such an inverted index data structure with DF and TF counts as required for the TF-IDF/cosine similarity.

When all records in the database(s) have been inserted into the inverted index, the TF and DF counts can be normalized and the inverse document frequencies (IDF) be calculated [37]. If Jaccard similarity is used neither frequency information nor normalization is required.

Once the inverted index data structure is built, overlapping clusters, called canopies, can be generated [29]. For this, initially all records are inserted into a pool of candidate records. A canopy cluster is created by randomly selecting a record r_c from this pool. This record will become the centroid of a new cluster. All records in the pool that are similar to r_c (according to the selected similarity measure) are added into the current cluster. The Jaccard similarity between r_c and any other record r_x in the pool is calculated as

$$s_J = \frac{|token(r_c) \cap token(r_x)|}{|token(r_c) \cup token(r_x)|}, \quad (17)$$

with the function $token(r)$ returning the tokens of the BKV of a record r, and $0 \leq s_J \leq 1$. When the TF-IDF/cosine similarity measure is used, the normalized TF and IDF weight values, as stored in the inverted index, are included into the similarity calculations [37], which makes this measure computationally more expensive. Once the similarities between r_c and all other records in the pool are calculated, an overlapping cluster can be created in two different ways: based on thresholds or nearest neighbors.

3.5.1 Threshold-Based Approach

In this originally proposed approach [29], [44], two similarity thresholds are used to create the overlapping clusters. All records r_x that are within a loose similarity, t_l, to r_c are inserted into the current cluster (e.g., all records with $t_l \leq s_J$). Of these, all records that are within a tight similarity threshold t_t (with $t_t \geq t_l$), will be removed from the pool of candidate records.

This process of randomly selecting a centroid record r_c, calculating the similarities between this and all other records in the pool, and inserting records into clusters, is repeated until no candidate records are left in the pool. If $t_l = t_t$, the clusters will not be overlapping, which means each record will be inserted into one cluster only. If both $t_l = 1$ and $t_t = 1$ (i.e., exact similarity only), canopy clustering will generate the same candidate record pairs as traditional blocking.

Estimating the number of candidate record pairs that will be generated with this approach is difficult (similar to q-gram based indexing), because this number depends upon the values of the thresholds t_l and t_t, the similarities between BKVs, and the frequency distribution of their tokens. Together, these factors determine how many record identifiers will be inserted into each cluster, and how many will be removed from the pool of candidate records in each step of the algorithm. The random selection of the records used as cluster centroids also results in a nondeterministic approach that can result in different clusters for each run, and thus different numbers of candidate record pairs generated.

One extreme situation would occur when the similarity values between all BKVs (calculated using their tokens as discussed above) are larger than the t_t threshold, resulting in one single cluster only that contains all record identifiers. The other extreme situation would occur when all BKVs are so different from each other that their similarities are below t_l, and thus each record is inserted only into the block that contains the record identifiers that have the same BKV. In this second situation, canopy clustering will generate the same candidate record pairs as traditional blocking.

Assuming the number of clusters (or blocks) generated equals b, and that all clusters contain the same number of record identifiers, then the number of candidate record pairs generated depends upon into how many clusters each record will be inserted. The smallest number of candidate

pairs will be generated when each record is inserted into one cluster only. On the other hand, if each record is inserted into v clusters, then, based on (1) and (12), the number of candidate record pairs generated for a record linkage can be estimated as

$$\frac{n_A n_B}{b} \leq u_{CCRL_T} \leq \frac{n_A n_B v^2}{b}, \quad (18)$$

and for a deduplication (using (2) and (13)) the estimated number of candidate record pairs generated is

$$\frac{n_A}{2}\left(\frac{n_A}{b}-1\right) \leq u_{CCD_T} \leq \frac{n_A v}{2}\left(\frac{n_A v}{b}-1\right). \quad (19)$$

As can be seen, for both a record linkage and a deduplication the upper bound depends quadratically on the number of times a record identifier is inserted into a cluster. Given that in reality the generated clusters will not be of uniform size, the largest clusters will generate the largest numbers of candidate record pairs (similar as with traditional blocking). For BKVs that follow a Zipf-like distribution, (3) and (4) can be extended with the overhead v similarly to (18) and (19) above.

3.5.2 Nearest Neighbor-Based Approach

An alternative to using two thresholds is to employ a nearest neighbor-based approach to create the overlapping clusters [39]. The idea is to replace the two threshold parameters, t_l and t_t, with two nearest neighbor parameters, n_l and n_t (with $n_l \geq n_t$). The first parameter, n_l, corresponds to the number of record identifiers that are inserted into each cluster, while n_t is the number of record identifiers that are removed from the pool of candidate records in each step of the algorithm.

Similar to the threshold-based approach, the process of creating overlapping clusters starts by randomly selecting a record r_c from the pool of initially all records. Similarities are then calculated between the r_c and the records r_x that have tokens in common in the inverted index. The n_l records closest to r_c are inserted into the current cluster, and of these the n_t records closest to r_c are removed from the pool.

This approach will result in all clusters containing n_l record identifiers, independently of the frequency distribution of the BKVs. Therefore, blocks of uniform size will be created, allowing the calculation of the number of generated record pairs. The number of clusters only depends upon the number of records in the database(s) to be matched or deduplicated, and the values of n_l and n_t. The number of clusters generated corresponds to n_A/n_t and n_B/n_t, respectively, and each cluster will contain n_l records. For a record linkage, the number of candidate record pairs generated therefore equals

$$u_{CCRL_N} = z_{CCRL_N} = \left(\frac{n_A n_l}{n_t}\right) \times \left(\frac{n_B n_l}{n_t}\right) = \frac{n_A n_B n_l^2}{n_t^2}, \quad (20)$$

while for a deduplication the number equals

$$u_{CCD_N} = z_{CCD_N} = \frac{n_A n_l}{2 n_t}\left(\frac{n_A n_l}{n_t}-1\right). \quad (21)$$

The drawback of this approach is similar to the drawback of the sorted neighborhood technique based on a sorted array, as discussed in Section 3.2.1. If there are BKVs that are frequent (like the surnames "Smith" or "Meier"), the generated clusters might not be big enough to include all records with these BKVs, and therefore true matches might be missed. The solution is to use BKVs that are the concatenation of several record fields and that have a large number of different values.

Compared to other indexing techniques, canopy clustering using both the threshold and the nearest neighbor approach is not sensitive to errors and variations at the beginning of BKVs, because the similarity measures used are independent of the order of where tokens appear in BKVs.

Previous experiments [39] have shown that using the nearest neighbor-based approach can result in an increase in the number of true matches in the candidate records pairs that are generated compared to the threshold-based approach, and also in a higher robustness of the canopy clustering technique with regard to changes in parameter settings. The experiments presented in Sections 4 and 5 will confirm these statements.

3.6 String-Map-Based Indexing

This indexing technique [45] is based on mapping BKVs (assumed to be strings) to objects in a multidimensional Euclidean space, such that the distances between pairs of strings are preserved. Any string similarity measure that is a distance function (such as edit-distance [35]) can be used in the mapping process. Groups of similar strings are then generated by extracting objects in this space that are similar to each other. The approach is based on a modification of the *FastMap* [46] algorithm, called *StringMap*, that has a linear complexity in the number of strings to be mapped [45].

The first step of string-map-based indexing iterates over d dimensions. For each dimension, the algorithm finds two pivot strings that are used to form orthogonal directions. Ideally, these two pivots are as far apart from each other as possible. To find the two pivot strings, an iterative farthest first selection process is used. Once the pivot strings have been selected for a dimension, the coordinates of all other strings are calculated based on the directions of these pivot strings. Selecting an appropriate dimension d is based on using a heuristic approach that iterates over a range of dimensions and selects the one that minimizes a cost function. Dimensions between 15 and 25 seem to achieve good results [45].

Once all strings are mapped into a multidimensional space using a suitable index data structure (the original implementation uses an R-tree [45]), in the second step of this indexing approach (the **Retrieve** step) clusters of similar objects (that refer to similar strings) are retrieved. In the implementation of string-map-based indexing evaluated in the experiments, the originally implemented R-tree data structure has been replaced with a grid-based index [47]. As reported, the performance of most tree-based index structures degrade rapidly with more than 15 to 20 dimensions [47], because nearly all objects in an index will be accessed when similarity searches are conducted. The grid-based index works by having a regular grid of dimensionality d implemented as an inverted index in each dimension. The index keys are the coordinate value of the objects, and all objects mapped into the same grid cell in a dimension are inserted into the same inverted index list.

Similar to canopy clustering-based indexing, overlapping clusters can be extracted from the multidimensional grid index. An object (referring to a BKV) is randomly picked from the pool of (initially all) objects in the grid-based index, and the objects in the same, as well as in neighboring grid cells, are retrieved from the index. Similar to canopy clustering, either two thresholds, t_l and t_t, or the number of nearest neighbors, n_l and n_t, can be used to insert similar objects into clusters, and remove objects from the pool with a similarity larger than t_t, or that are the n_t nearest objects to the centroid object. Equations (18) to (21) can be used to estimate the number of record pairs that will be generated with string-map-based indexing and using either a threshold or a nearest neighbor-based approach.

A variation of this mapping-based indexing technique has recently been proposed [48], with the basic idea being to first map records into a multi-dimensional space, followed by a mapping into a second lower-dimensional metric space where edit-distance calculations are performed. Using a KD-tree and a nearest neighbor-based similarity approach allows for efficient matching. Experiment showed a reduction in runtime of 30 to 60 percent compared to string-map-based indexing, while at the same time keeping the matching accuracy [48].

4 Experimental Evaluation

The aim of the experiments conducted was to evaluate the efficiency and performance of the presented indexing techniques within a common framework, in order to answer questions such as: how do parameter values and the choice of the blocking key influence the number and quality of the candidate record pairs generated? How do different indexing techniques perform with different types of data? Which indexing techniques show better scalability to larger databases?

All presented indexing techniques were implemented in Python within the *Febrl* open source record linkage system [26]. (available from: https://sourceforge.net/projects/febrl/) To facilitate repeatability of the presented results, the evaluation program used for these experiments (evalIndexing.py) will be published as part of the next version of *Febrl*. All experiments were conducted on an otherwise idle compute server with two 2.33 GHz quad-core CPUs and 16 Gigabytes of main memory, running Linux 2.6.32 (Ubuntu 10.04) and using Python 2.6.5.

4.1 Test Data Sets

Two series of experiments were conducted, the first using four "real" data sets that have previously been used by the record linkage research community, and the second using artificial data sets. Table 3 summarizes these data sets. The aim of the first series of experiments was to investigate how different indexing techniques are able to handle various types of data, while the second series was aimed at investigating the scalability of the different indexing techniques to larger data sets.

The first three "real" data sets were taken from the *SecondString* toolkit.[1] "Census" contains records that were generated by the US Census Bureau based on real census data; "Cora" contains bibliographic records of machine learning publications; and "Restaurant" contains records extracted from the Fodor and Zagat restaurant guides. The "CDDB" data set contains records of audio CDs, such as their title, artist, genre, and year. This last data set was recently used in the evaluation of a novel indexing technique [41]. The true match status of all record pairs is available in all four of these data sets.

TABLE 3
Data Sets Used in Experiments

Data set name	Task	Number of records	Total number of true matches
Census	Linkage	449 + 392	327
Restaurant	Deduplication	864	112
Cora	Deduplication	1,295	17,184
CDDB	Deduplication	9,763	607
Clean	Linkage	1,000–100,000	200–20,000
Dirty	Linkage	1,000–100,000	400–40,000

Artificial data sets containing 1,000, 5,000, 10,000, 50,000, and 100,000 records, respectively, were generated.

Artificial data sets were generated using the *Febrl* data generator [49]. This generator first creates *original* records based on frequency tables that contain real name and address values, as well as other personal attributes; followed by the generation of *duplicates* of these records based on random modifications such as inserting, deleting, or substituting characters, and swapping, removing, inserting, splitting, or merging words. The types and frequencies of these modifications are also based on real characteristics. The true match status of all record pairs is known. The original and duplicate records were then stored into one file each to facilitate their linkage.

As shown in Table 3, two series of artificial data sets were created. The "Clean" data contain 80 percent original and 20 percent duplicate records, with up to three duplicates for one original record, a maximum of one modification per attribute, and a maximum of three modifications per record. The "Dirty" data contain 60 percent original and 40 percent duplicate records, with up to nine duplicates per original record, a maximum of three modifications per attribute, and a maximum of 10 modifications per record.

4.2 Quality and Complexity Measures

Four measures are used to assess the complexity of the indexing step and the quality of the resulting candidate record pairs [9], [10]. The total number of matched and nonmatched record pairs are denoted with n_M and n_N, respectively, with $n_M + n_M = n_A \times n_B$ for the linkage of two databases, and $n_M + n_N = n_A(n_A - 1)/2$ for the deduplication of one database. The number of true matched and true nonmatched record pairs generated by an indexing technique is denoted with s_M and s_N, respectively, with $s_M + s_N \leq n_M + n_N$.

The reduction ratio, $RR = 1.0 - \frac{s_M + s_N}{n_M + n_N}$, measures the reduction of the comparison space, i.e., the fraction of record pairs that are removed by an indexing technique. The higher the RR value, the less candidate record pairs are being generated. However, reduction ratio does not take the quality of the generated candidate record pairs into account (how many are true matches or not).

1. Available from: http://secondstring.sourceforge.net.

TABLE 4
The Labels Used in the Result Figures, the Number of Different Parameter Settings Evaluated, and the Runtimes in Milliseconds Per Candidate Record Pair Required to Build Each of the Evaluated Indexing Techniques

Indexing technique	Label used in figures	Number of settings	Time in milli-seconds per candidate record pair			
			Minimum	Median	Average	Maximum
Traditional blocking	TBlo	1	0.002	0.591	0.511	0.972
Array based sorted neighbourhood	SorAr	5	0.011	0.059	0.081	0.288
Inverted index based sorted neighbourhood	SorII	5	0.002	0.033	0.293	3.040
Adaptive sorted neighbourhood	AdSor	8	0.002	0.952	1.128	4.702
Q-gram based indexing	QGr	4	0.005	4.118	1,170.716	163,484.394
Threshold based canopy clustering	CaTh	8	0.003	4.252	18.194	380.214
Nearest neighbour based canopy clustering	CaNN	8	0.004	0.151	1.912	39.190
Threshold based string-map indexing	STMTh	32	0.004	0.488	21.715	664.862
Nearest neighbour based string-map indexing	StMNN	32	0.018	2.045	19.386	695.101
Suffix-array based indexing	SuAr	6	0.024	1.119	11.498	168.561
Suffix-array based indexing using all sub-strings	SuArSu	6	0.017	3.542	28.082	438.191
Robust suffix-array indexing	RoSuA	48	0.010	0.434	0.856	10.421

Pairs completeness, $PC = \frac{s_M}{n_M}$, is the number of true matched record pairs generated by an indexing technique divided by the total number of true matched pairs. It measures how effective an indexing technique is in not removing true matched record pairs. PC corresponds to *recall* as used in information retrieval [37].

Finally, pairs quality, $PQ = \frac{s_M}{s_M + s_N}$, is the number of true matched record pairs generated by an indexing technique divided by the total number of record pairs generated. A high PQ value means an indexing technique is efficient and generates mostly true matched record pairs. On the other hand, a low PQ value means a large number of nonmatches are also generated. PQ corresponds to *precision* as used in information retrieval. The f-score (or f-measure) [9], the harmonic mean of PC and PQ, $f = \frac{PC * PQ}{PC + PQ}$, will also be reported.

4.3 Experimental Setup

Rather than trying to find optimal parameter settings for each combination of blocking key definition, indexing technique and test data set, a large number of settings were evaluated to provide a better understanding of the average performance and scalability of the different indexing techniques on different data sets. Because in many real-world applications no training data are available that would allow optimal parameter tuning, domain and record linkage experts are commonly tasked with finding the best settings experimentally.

For each data set, three different blocking keys were defined using a variety of combinations of record fields. String fields such as names and addresses were phonetically encoded using the Double-Metaphone [35] algorithm. For example, for the "Census" data set, one blocking key definition consisted of encoded surnames concatenated with initials and zipcodes, while a second consisted of encoded given names concatenated with encoded suburb names. Due to space limitation, not all blocking key definitions can be described in detail.

A large variety of parameter settings were evaluated. Four string similarity functions (*Jaro-Winkler*, *bigram*, *edit-distance*, and *longest common substring*) [35] were employed for the adaptive sorted neighborhood, the robust suffix array, and the string-map-based indexing techniques. For the two nonadaptive sorted neighborhood techniques, the window size was set to $w = \{2, 3, 5, 7, 10\}$. Similarity thresholds were set to $t = \{0.8, 0.9\}$ and q-grams to $q = \{2, 3\}$ for all indexing techniques that require these parameters. For suffix array-based indexing, the minimum suffix length and maximum block sizes were set to $l_m = \{3, 5\}$ and $b_M = \{5, 10, 20\}$. For canopy clustering, both the *Jaccard* and *TF-IDF/cosine* similarities were used, in combination with global thresholds $t_t/t_l = \{0.9/0.8, 0.8/0.7\}$ or nearest neighbor parameters $n_t/n_l = \{5/10, 10/20\}$. The same threshold and nearest values were also used for string-map-based indexing. The grid size for this technique was set to 100 and 1,000, and the mapping dimension to $d = \{15, 20\}$.

For each data set, a total of 163 parameter settings were evaluated. Table 4 summarizes the experimental setup and shows runtime results. Figs. 9, 10, 11, 12, and 13 show for each indexing technique the average and standard deviation values over all blocking key definitions and combinations of parameter settings. The presented results on these various data sets should therefore provide some indication of the performance and scalability of the different indexing techniques. Note that for the scalability experiments not all results are shown, because either an indexing technique required more than the available 16 Gigabytes of main memory, or its runtime was prohibitively slow to conduct many experiments.

5 Discussion

Looking at the runtime results shown in the right-hand side of Table 4, one can clearly see that the q-gram-based indexing technique is the overall slowest technique by a very large margin (in average 40 times slower than the second slowest technique). This confirms earlier experiments [32], and as a result this technique is not suitable for linking or deduplicating large databases.

Both string-map-based indexing approaches, the suffix array-based approaches, and threshold-based canopy clustering also have fairly slow average and maximum runtimes. On the other hand, the more simpler approaches, like traditional blocking and the array-based sorted neighborhood approach, are the overall fastest techniques. Among the other fastest techniques are the robust suffix array and adaptive sorted neighborhood approaches. In the following discussion we will see if these fast indexing times

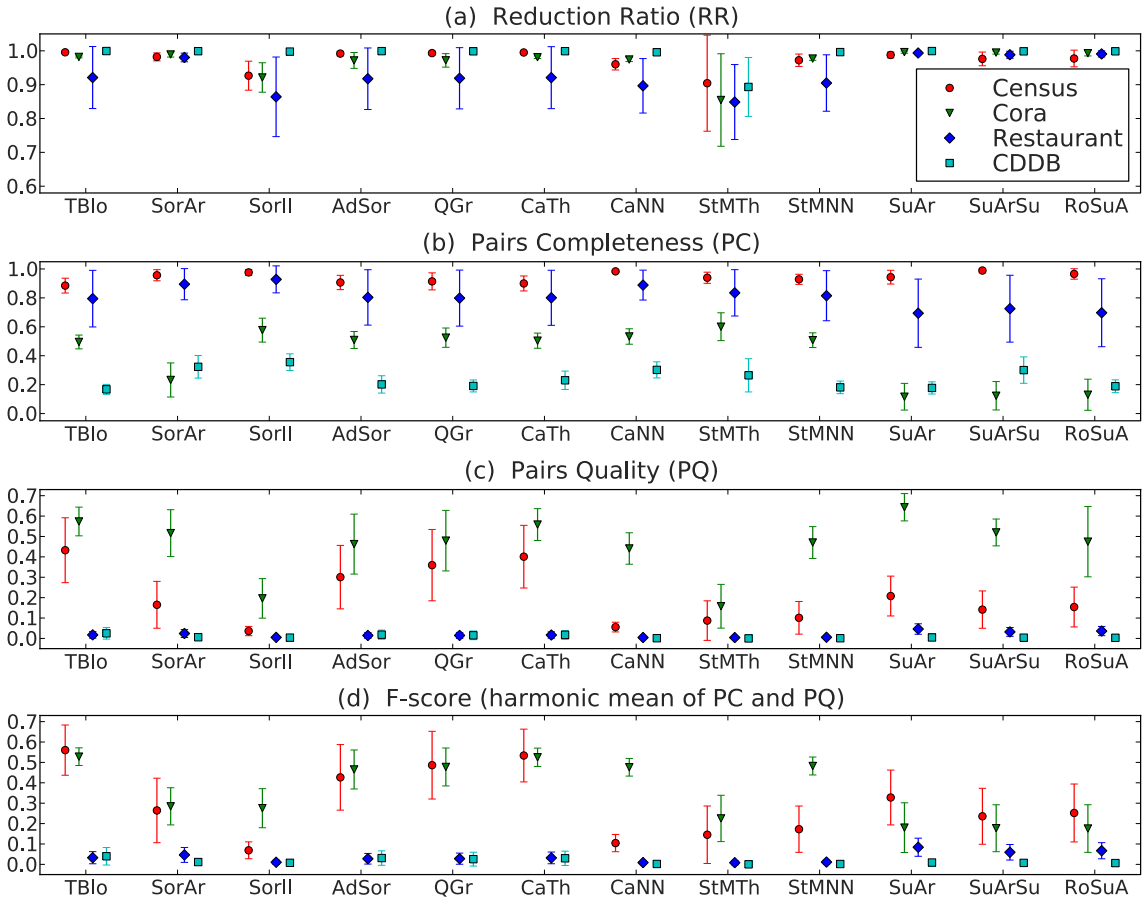

Fig. 9. Experimental results for the four "real" data sets. Average values and standard deviations are shown.

come at the cost of lower indexing quality (i.e., lower PC and PQ values).

Fig. 9 shows the results for the four "real" data sets. As can be seen in Fig. 9a, some indexing techniques have large variations in the reduction ratio (RR) they attain, while others have a high RR independently of the data sets they are applied on. The array-based sorted neighborhood and the three suffix array-based indexing techniques achieve nearly uniformly high RR values for all data sets, because the size of the blocks generated by them is either independent of the data to be linked or deduplicated, or limited to a maximum size determined by a parameter. Therefore, because the number of candidate record pairs generated by these techniques can easily be calculated, they can be useful for applications where a linkage or deduplication must be done within a certain amount of time.

On the other hand, the large variations of RR values by other indexing techniques for some data sets are due to the varying sizes of the blocks generated by them. For these techniques, block sizes depend upon the frequency distribution of the BKVs.

The quality of the candidate record pairs generated by the different indexing techniques, measured using PC, PQ, and the f-score, is mostly influenced by the characteristics of the data set and the choice of blocking key definition. This can be seen by the large standard deviations for some data sets in Figs. 9b, 9c, and 9d. All three measures differ more prominently between data sets than between indexing techniques. For the "Census" data set, for example, all techniques achieve a PC value above 0.8, while for the "CDDB" data set none produces a PC value larger than 0.4. This highlights the need for the careful definition of blocking keys, which needs to be done uniquely to each data set. As the very low PQ and f-score results for "CDDB" and "Restaurant" show, due to variations and errors in these data sets, that cannot be overcome with appropriate blocking key definitions, it might not be possible at all to achieve good quality indexing with traditional techniques.

Among the techniques that attain the lowest PC and f-score values are the suffix array-based approaches. This is because the high RR they achieve comes at a cost of low PQ and f-score values. The highest performing technique with regard to PC is the inverted index-based sorted neighborhood approach, which is surprising given its sensitivity to errors and variations at the beginning of BKVs.

The overall efficiency of the different indexing techniques is shown in Fig. 9d as the f-score of PC and PQ. Surprisingly, traditional blocking is the best performing technique for two of the four data sets, closely followed by threshold-based canopy clustering, q-gram-based indexing, and the adaptive sorted neighborhood technique. This figure once more highlights that the definition of suitable blocking keys is one of the most crucial components in the indexing step for record linkage or deduplication, and not the actual indexing technique employed.

Moving on to the results achieved with the artificial data sets shown in Figs. 10, 11, 12, and 13, as can be seen the RR

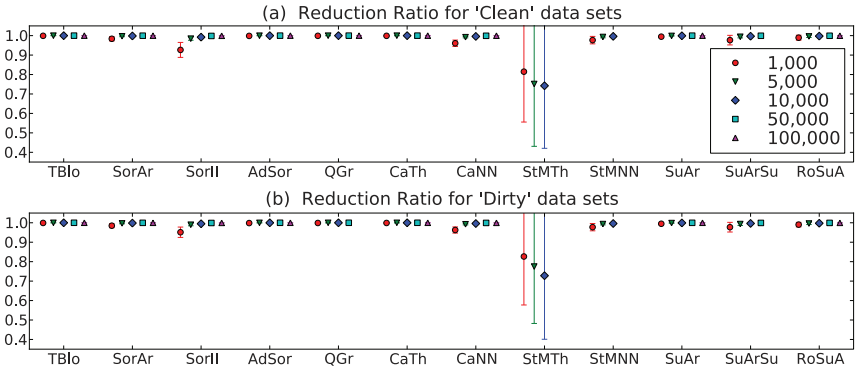

Fig. 10. Reduction ratio results for the artificial data sets. Average values and standard deviations are shown.

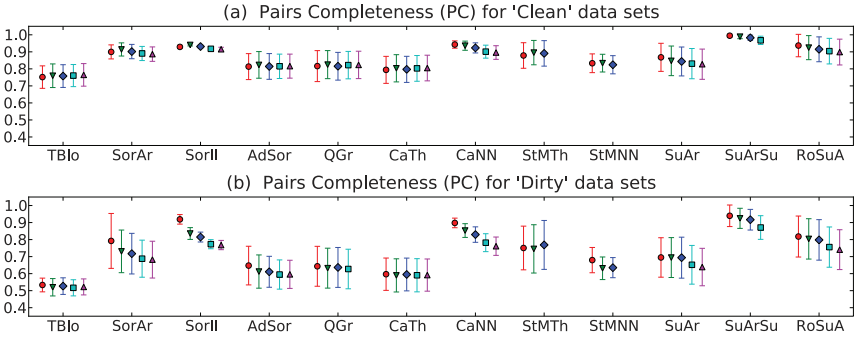

Fig. 11. Pairs completeness results for the artificial data sets. Average values and standard deviations are shown.

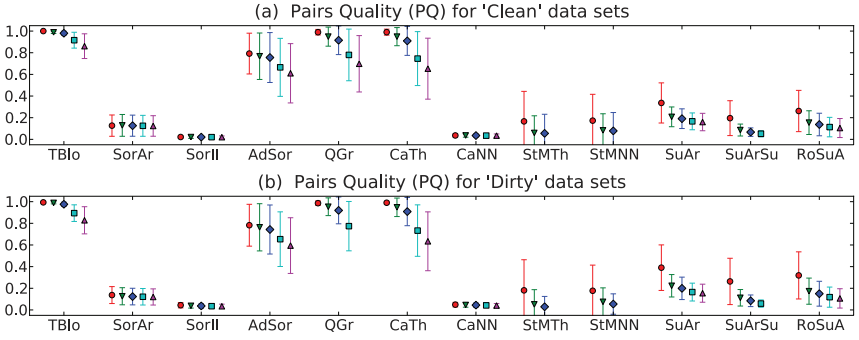

Fig. 12. Pairs quality results for the artificial data sets. Average values and standard deviations are shown.

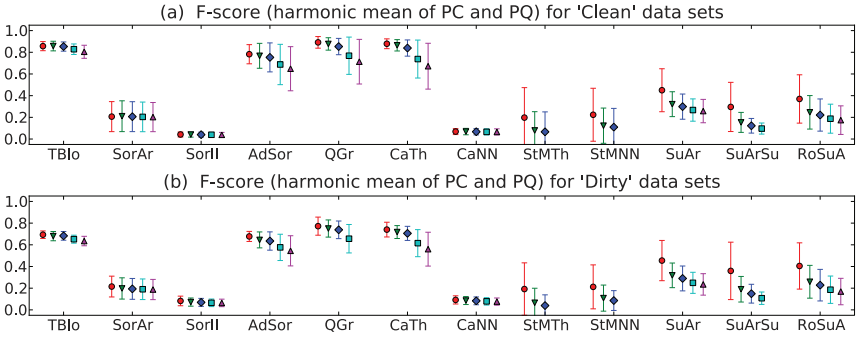

Fig. 13. F-score results for the synthetic data sets. Average values and standard deviations are shown.

values for most indexing techniques stay high—or get even higher—as the data sets get larger. This indicates a subquadratic increase in the number of candidate record pairs generated as the data sets get larger. One exception is the threshold-based string-map indexing technique, which not only has the lowest average RR values in general, but also shows to be very sensitive toward parameter settings.

As expected, the PC values are higher for the "Clean" data sets compared to the "Dirty" data sets, because the BKVs for the former contain less errors and variations and thus more records were inserted into the correct blocks. As Figs. 11 and 12 show, for several indexing techniques the PC or PQ values drop significantly as the data sets get larger. As can be seen, having a constant PC value across data sets size comes at the cost of lower PQ values, and vice versa.

The reason for the drop in PC values are the fixed window size for the sorted neighborhood approaches, the fixed number of nearest neighbors for the canopy clustering and string-map-based approaches, and the fixed maximum block size for the suffix array-based techniques. As the data sets get larger, more records will have the same or similar BKVs, and the fixed block size limits will result in records that do have the same (or very similar BKVs) not being included into the same block, thereby missing increasingly more true matches with larger data sets. On the other hand, traditional blocking and the threshold-based indexing techniques do not have maximum block sizes, and thus all records with the same or similar BKVs are inserted into the same blocks without limitations.

The costs of being able to keep constant PC values with larger data sets are lower PQ values. This means that as data sets get larger, the number of candidate record pairs that will be generated increases faster for threshold-based techniques than for techniques that somehow limit the maximum block size. This tradeoff between PC and PQ is similar to the precision-recall tradeoff in information retrieval [37].

One aspect of indexing techniques that is of importance to their practical use is their robustness with regard to parameter settings. Ideally, an indexing technique should achieve a high RR and a high f-score value for a large variety of parameter settings, because otherwise a user needs to carefully tune the parameters of an indexing technique. As Fig. 13 shows, the string-map and suffix array-based approaches have the largest standard deviations in the f-score values they achieve.

For effective parameter tuning, some form of "gold standard" data, where the true match status of record pairs is known, must be available. Such data must have the same characteristics as the data to be linked or deduplicated. As can be seen from Fig. 13, traditional blocking, the adaptive sorted neighborhood approach and threshold-based canopy clustering achieve high f-score values for all their parameter settings.

6 Related Research

Research into indexing for record linkage and deduplication can be classified into two categories. The first category is to develop new and improve existing techniques with the aim of making them more scalable to large data sets while keeping a high linkage quality [22], [33], [40], [41], [43], [48], [50], [51]. The techniques presented in this survey are efforts toward this goal.

The second category of research into indexing is the development of techniques that can learn optimal blocking key definitions. Traditionally, the choice of blocking keys is made manually by domain and record linkage experts. Recently, two supervised machine learning-based approaches to optimally select blocking keys have been proposed [52], [53]. They either employ predicate-based formulations of learnable blocking functions [52], or use the sequential covering algorithm which discovers disjunctive sets of rules [53]. Both approaches aim to define blocking keys such that the number of true matches in the candidate record pairs is maximized, while keeping the total number of candidate pairs as small as possible. Both approaches rely on training examples, i.e., pairs of true matches and nonmatches. Such training data are often not available in real-world applications, or they have to be prepared manually. In principle, these learning approaches can be employed with any of the indexing techniques presented in this survey.

Another approach to reduce the computational efforts of the record linkage process is to minimize the number of costly (string) comparisons that need to be made between records. One recently proposed technique is based on a matching tree that allows early match decisions without having to compare all attributes between two records [54], while another technique sequentially assesses the attributes and stops the comparison of two records once a match decision can be made [55].

A large amount of work has also been conducted by the database community on similarity joins [42], [51], [56], [57], [58], [59], where the aim is to facilitate efficient and scalable approximate joins for similarity measures such as edit or Jaccard distance. Another avenue of work is to efficiently index uncertain databases [60], as well as finding similarities between objects in uncertain databases [61]. Concurrently, the information retrieval community has developed techniques to detect duplicate documents returned by (web) search queries [6], [7]. In this domain, fast matching and scalability to very large data collections are of paramount importance.

Real-time entity resolution of databases that contain personal information has recently also attracted some interest [21], because many applications increasingly require the matching of query records to large databases of known entities in real time rather than in batch mode.

7 Conclusions and Future Work

This paper has presented a survey of six indexing techniques with a total of 12 variations of them. The number of candidate record pairs generated by these techniques has been estimated theoretically, and their efficiency and scalability has been evaluated using various data sets. These experiments highlight that one of the most important factors for efficient and accurate indexing for record linkage and deduplication is the proper definition of blocking keys. Because training data in the form of true matches and true nonmatches is often not available in real-world applications, it is commonly up to domain and linkage experts to decide how such blocking keys are defined.

The experimental results showed that there are large differences in the number of true matched candidate record pairs generated by the different techniques, but also large differences for several indexing techniques depending upon the setting of their parameters. The variety of parameters that have to be set by a user, and the sensitivity of some of them (especially global thresholds) with regard to the candidate record pairs generated, makes it somewhat difficult to successfully apply these techniques in practice, as parameter settings depend both upon the quality and characteristics of the data to be linked or deduplicated.

Due to space limitation it was not possible to include an empirical evaluation of the theoretical estimates of the number of candidate record pairs that will be generated, as was provided in Sections 3.1 to 3.6. Such an evaluation will be part of future work. Other future work includes the implementation of further recently developed new indexing

techniques [22], [41], [48] into the *Febrl* framework, as well as the investigation of learning techniques for efficient and accurate indexing [52], [53].

The indexing techniques presented in this survey are heuristic approaches that aim to split the records in a database (or databases) into (possibly overlapping) blocks such that matches are inserted into the same block and nonmatches into different blocks. While future work in the area of indexing for record linkage and deduplication should include the development of more efficient and more scalable new indexing techniques, the ultimate goal of such research will be to develop techniques that generate blocks such that it can be proven that 1) all comparisons between records within a block will have a certain minimum similarity with each other, and 2) the similarity between records in different blocks is below this minimum similarity.

REFERENCES

[1] W.E. Winkler, "Methods for Evaluating and Creating Data Quality," *Elsevier Information Systems,* vol. 29, no. 7, pp. 531-550, 2004.

[2] D.E. Clark, "Practical Introduction to Record Linkage for Injury Research," *Injury Prevention,* vol. 10, pp. 186-191, 2004.

[3] C.W. Kelman, J. Bass, and D. Holman, "Research Use of Linked Health Data—A Best Practice Protocol," *Australian NZ J. Public Health,* vol. 26, pp. 251-255, 2002.

[4] W.E. Winkler, "Overview of Record Linkage and Current Research Directions," Technical Report RR2006/02, US Bureau of the Census, 2006.

[5] J. Jonas and J. Harper, "Effective Counterterrorism and the Limited Role of Predictive Data Mining," *Policy Analysis,* no. 584, pp. 1-11, 2006.

[6] H. Hajishirzi, W. Yih, and A. Kolcz, "Adaptive Near-Duplicate Detection via Similarity Learning," *Proc. 33rd Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR '10),* pp. 419-426, 2010.

[7] W. Su, J. Wang, and F.H. Lochovsky, "Record Matching over Query Results from Multiple Web Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 22, no. 4, pp. 578-589, Apr. 2010.

[8] M. Bilenko, S. Basu, and M. Sahami, "Adaptive Product Normalization: Using Online Learning for Record Linkage in Comparison Shopping," *Proc. IEEE Int'l Conf. Data Mining (ICDM '05),* pp. 58-65, 2005.

[9] P. Christen and K. Goiser, "Quality and Complexity Measures for Data Linkage and Deduplication," *Quality Measures in Data Mining,* ser. Studies in Computational Intelligence, F. Guillet and H. Hamilton, eds., vol. 43, Springer, pp. 127-151, 2007.

[10] M.G. Elfeky, V.S. Verykios, and A.K. Elmagarmid, "TAILOR: A Record Linkage Toolbox," *Proc. 18th Int'l Conf. Data Eng. (ICDE '02),* 2002.

[11] I.P. Fellegi and A.B. Sunter, "A Theory for Record Linkage," *J. Am. Statistical Soc.,* vol. 64, no. 328, pp. 1183-1210, 1969.

[12] W.W. Cohen, P. Ravikumar, and S. Fienberg, "A Comparison of String Distance Metrics for Name-Matching Tasks," *Proc. Workshop Information Integration on the Web (IJCAI '03),* 2003.

[13] W.W. Cohen, "Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '98),* pp. 201-212, 1998.

[14] H. Galhardas, D. Florescu, D. Shasha, and E. Simon, "An Extensible Framework for Data Cleaning," *Proc. 16th Int'l Conf. Data Eng. (ICDE '00),* 2000.

[15] E. Rahm and H.H. Do, "Data Cleaning: Problems and Current Approaches," *IEEE Technical Committee Data Eng. Bull.,* vol. 23, no. 4, pp. 3-13, Dec. 2000.

[16] J.I. Maletic and A. Marcus, "Data Cleansing: Beyond Integrity Analysis," *Proc. Fifth Conf. Information Quality (IQ '00),* pp. 200-209, 2000.

[17] M. Bilenko and R.J. Mooney, "On Evaluation and Training-Set Construction for Duplicate Detection," *Proc. Workshop Data Cleaning, Record Linkage and Object Consolidation (SIGKDD '03),* pp. 7-12, 2003.

[18] A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios, "Duplicate Record Detection: A Survey," *IEEE Trans. Knowledge and Data Eng.,* vol. 19, no. 1, pp. 1-16, Jan. 2007.

[19] A. Aizawa and K. Oyama, "A Fast Linkage Detection Scheme for Multi-Source Information Integration," *Proc. Int'l Workshop Challenges in Web Information Retrieval and Integration (WIRI '05),* 2005.

[20] I. Bhattacharya and L. Getoor, "Collective Entity Resolution in Relational Data," *ACM Trans. Knowledge Discovery from Data,* vol. 1, no. 1, pp. 5-es, 2007.

[21] P. Christen, R. Gayler, and D. Hawking, "Similarity-Aware Indexing for Real-Time Entity Resolution," *Proc. 18th ACM Conf. Information and Knowledge Management (CIKM '09),* pp. 1565-1568, 2009.

[22] S.E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina, "Entity Resolution with Iterative Blocking," *Proc. 35th ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '09),* pp. 219-232, 2009.

[23] X. Dong, A. Halevy, and J. Madhavan, "Reference Reconciliation in Complex Information Spaces," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '05),* pp. 85-96, 2005.

[24] M.A. Hernandez and S.J. Stolfo, "The Merge/Purge Problem for Large Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '95),* 1995.

[25] T. Churches, P. Christen, K. Lim, and J.X. Zhu, "Preparation of Name and Address Data for Record Linkage Using Hidden Markov Models," *BioMed Central Medical Informatics and Decision Making,* vol. 2, no. 9, 2002.

[26] P. Christen, "Febrl: An Open Source Data Cleaning, Deduplication and Record Linkage System With a Graphical User Interface," *Proc. 14th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '08),* pp. 1065-1068, 2008.

[27] L. Gu and R. Baxter, "Decision Models for Record Linkage," Selected Papers from AusDM, LNCS 3755, Springer, 2006.

[28] P. Christen, "Automatic Record Linkage Using Seeded Nearest Neighbour and Support Vector Machine Classification," *Proc. 14th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '08),* pp. 151-159, 2008.

[29] W.W. Cohen and J. Richman, "Learning to Match and Cluster Large High-Dimensional Data Sets for Data Integration," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '02),* pp. 475-480, 2002.

[30] S. Sarawagi and A. Bhamidipaty, "Interactive Deduplication Using Active Learning," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '02),* 2002.

[31] S. Tejada, C.A. Knoblock, and S. Minton, "Learning Domain-Independent String Transformation Weights for High Accuracy Object Identification," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '02),* 2002.

[32] R. Baxter, P. Christen, and T. Churches, "A Comparison of Fast Blocking Methods for Record Linkage," *Proc. ACM Workshop Data Cleaning, Record Linkage and Object Consolidation (SIGKDD '03),* pp. 25-27, 2003.

[33] J. Nin, V. Muntes-Mulero, N. Martinez-Bazan, and J.-L. Larriba-Pey, "On the Use of Semantic Blocking Techniques for Data Cleansing and Integration," *Proc. 11th Int'l Database Eng. and Applications Symp. (IDEAS '07),* 2007.

[34] M.A. Hernandez and S.J. Stolfo, "Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem," *Data Mining and Knowledge Discovery,* vol. 2, no. 1, pp. 9-37, 1998.

[35] P. Christen, "A Comparison of Personal Name Matching: Techniques and Practical Issues," *Proc. IEEE Sixth Data Mining Workshop (ICDM '06),* 2006.

[36] K. Goiser and P. Christen, "Towards Automated Record Linkage," *Proc. Fifth Australasian Conf. Data Mining and Analystics (AusDM '06),* vol. 61, pp. 23-31, 2006.

[37] I.H. Witten, A. Moffat, and T.C. Bell, *Managing Gigabytes,* second ed. Morgan Kaufmann, 1999.

[38] M. Harada, S. Sato, and K. Kazama, "Finding Authoritative People from the Web," *Proc. ACM/IEEE-CS Joint Conf. Digital Libraries,* pp. 306-313, 2004.

[39] P. Christen, "Towards Parameter-Free Blocking for Scalable Record Linkage," Technical Report TR-CS-07-03, Dept. of Computer Science, The Australian Nat'l Univ., 2007.

[40] S. Yan, D. Lee, M.Y. Kan, and L.C. Giles, "Adaptive Sorted Neighborhood Methods for Efficient Record Linkage," *Proc. Seventh ACM/IEEE-CS Joint Conf. Digital Libraries (JCDL '07),* 2007.

[41] U. Draisbach and F. Naumann, "A Comparison and Generalization of Blocking and Windowing Algorithms for Duplicate Detection," *Proc. Workshop Quality in Databases (VLDB '09)*, 2009.
[42] L. Gravano, P.G. Ipeirotis, H.V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate String Joins in a Database (Almost) for Free," *Proc. 27th Int'l Conf. Very Large Data Bases (VLDB '01)*, pp. 491-500, 2001.
[43] T. de Vries, H. Ke, S. Chawla, and P. Christen, "Robust Record Linkage Blocking Using Suffix Arrays," *Proc. ACM Conf. Information and Knowledge Management (CIKM '09)*, pp. 305-314. 2009.
[44] A. McCallum, K. Nigam, and L.H. Ungar, "Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching," *Proc. Sixth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '00)*, pp. 169-178, 2000.
[45] L. Jin, C. Li, and S. Mehrotra, "Efficient Record Linkage in Large Data Sets," *Proc. Eighth Int'l Conf. Database Systems for Advanced Applications (DASFAA '03)*, pp. 137-146, 2003.
[46] C. Faloutsos and K.-I. Lin, "Fastmap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '95)*, pp. 163-174, 1995.
[47] C.C. Aggarwal and P.S. Yu, "The IGrid Index: Reversing the Dimensionality Curse for Similarity Indexing in High Dimensional Space," *Proc. Sixth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '00)*, pp. 119-129, 2000.
[48] N. Adly, "Efficient Record Linkage Using a Double Embedding Scheme," *Proc. Int'l Conf. Data Mining (DMIN '09)*, pp. 274-281, 2009.
[49] P. Christen and A. Pudjijono, "Accurate Synthetic Generation of Realistic Personal Information," *Proc. 13th Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining (PAKDD '09)*, vol. 5476, pp. 507-514, 2009.
[50] T. de Vries, H. Ke, S. Chawla, and P. Christen, "Robust Record Linkage Blocking Using Suffix Arrays and Bloom Filters," *ACM Trans. Knowledge Discovery from Data*, vol. 5, no. 2, pp. 1-27, 2011.
[51] M. Weis, F. Naumann, U. Jehle, J. Lufter, and H. Schuster, "Industry-Scale Duplicate Detection," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1253-1264, 2008.
[52] M. Bilenko, B. Kamath, and R.J. Mooney, "Adaptive Blocking: Learning to Scale up Record Linkage," *Proc. Sixth Int'l Conf. Data Mining (ICDM '06)*, pp. 87-96, 2006.
[53] M. Michelson and C.A. Knoblock, "Learning Blocking Schemes for Record Linkage," *Proc. 21st Nat'l Conf. Artificial Intelligence (AAAI '06)*, 2006.
[54] D. Dey, V. Mookerjee, and D. Liu, "Efficient Techniques for Online Record Linkage," *IEEE Trans. Knowledge and Data Eng.*, vol. 23, no. 3, pp. 373-387, Mar. 2011.
[55] G.V. Moustakides and V.S. Verykios, "Optimal Stopping: A Record-Linkage Approach," *J. Data and Information Quality*, vol. 1, pp. 9:1-9:34, 2009.
[56] A. Behm, S. Ji, C. Li, and J. Lu, "Space-Constrained Gram-Based Indexing for Efficient Approximate String Search," *Proc. IEEE Int'l Conf. Data Eng. (ICDE '09)*, pp. 604-615, 2009.
[57] N. Koudas, A. Marathe, and D. Srivastava, "Flexible String Matching against Large Databases in Practice," *Proc. 13th Int'l Conf. Very Large Data Bases (VLDB '04)*, pp. 1086-1094, 2004.
[58] S. Sarawagi and A. Kirpal, "Efficient Set Joins on Similarity Predicates," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '04)*, pp. 754-765, 2004.
[59] C. Xiao, W. Wang, and X. Lin, "Ed-Join: An Efficient Algorithm for Similarity Joins with Edit Distance Constraints," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 933-944, 2008.
[60] Y. Zhang, X. Lin, W. Zhang, J. Wang, and Q. Lin, "Effectively Indexing the Uncertain Space," *IEEE Trans. Knowledge and Data Eng.*, vol. 22, no. 9, pp. 1247-1261, Sept. 2010.
[61] T. Bernecker, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Zuefle, "Scalable Probabilistic Similarity Ranking in Uncertain Databases," *IEEE Trans. Knowledge and Data Eng.*, vol. 22, no. 9, pp. 1234-1246, Sept. 2010.

Peter Christen received the diploma degree in computer science engineering from ETH Zürich in 1995 and the PhD degree in computer science from the University of Basel in 1999 (both in Switzerland). His research interests include data mining and data matching (record linkage). He has published more than 50 papers in these areas, and he is the principle developer of the *Febrl* (Freely Extensible Biomedical Record Linkage) open source data cleaning, deduplication, and record linkage system.