Duplicate Record Detection: A Survey

Ahmed K. Elmagarmid
Purdue University

Panagiotis G. Ipeirotis
New York University

Vassilios S. Verykios
University of Thessaly

Abstract

Often, in the real world, entities have two or more representations in databases. Duplicate records do not share a common key and/or they contain errors that make duplicate matching a difficult task. Errors are introduced as the result of transcription errors, incomplete information, lack of standard formats or any combination of these factors. In this article, we present a thorough analysis of the literature on duplicate record detection. We cover similarity metrics that are commonly used to detect similar field entries, and we present an extensive set of duplicate detection algorithms that can detect approximately duplicate records in a database. We also cover multiple techniques for improving the efficiency and scalability of approximate duplicate detection algorithms. We conclude with a coverage of existing tools and with a brief discussion of the big open problems in the area.

Index Terms

duplicate detection, data cleaning, data integration, record linkage, data deduplication, instance identification, database hardening, name matching, identity uncertainty, entity resolution, fuzzy duplicate detection, entity matching

I. INTRODUCTION

Databases play an important role in today's IT based economy. Many industries and systems depend on the accuracy of databases to carry out operations. Therefore, the quality of the information (or the lack thereof) stored in the databases, can have significant cost implications to a system that relies on information to function and conduct business. In an error-free system with perfectly clean data, the construction of a comprehensive view of the data consists of linking –in relational terms, joining– two or more tables on their key fields. Unfortunately, data often lack a unique, global identifier that would permit such an operation. Furthermore, the data are neither carefully controlled for quality nor defined in a consistent way across different data sources. Thus, data quality is often compromised by many factors, including data entry errors (e.g., *Microsft* instead of *Microsoft*), missing integrity constraints (e.g., allowing entries such as *EmployeeAge=567*), and multiple conventions for recording information (e.g., *44 W. 4th St.* vs. *44 West Fourth Street*). To make things worse, in independently managed databases not only the values, but the structure, semantics and underlying assumptions about the data may differ as well.

Often, while integrating data from different sources to implement a data warehouse, organizations become aware of potential systematic differences or conflicts. Such problems fall under the umbrella-term *data heterogeneity* [14]. *Data cleaning* [77], or *data scrubbing* [96], refer to the process of resolving such identification problems in the data. We distinguish between two types of data heterogeneity: *structural* and *lexical*. *Structural heterogeneity* occurs when the fields of the tuples in the database are structured differently in different databases. For example, in one database, the customer address might be recorded in one field named, say, *addr*, while in another database the same information might be stored in multiple fields such as *street*, *city*, *state*, and *zipcode*. *Lexical heterogeneity* occurs when the tuples have identically structured fields across databases, but the data use different representations to refer to the same real-world object (e.g., *StreetAddress=44 W. 4th St.* vs. *StreetAddress=44 West Fourth Street*).

In this paper, we focus on the problem of lexical heterogeneity and survey various techniques which have been developed for addressing this problem. We focus on the case where the input is a set of *structured* and *properly segmented* records, i.e., we focus mainly on cases of database records. Hence, we do not cover solutions for the various other problems, such that of *mirror detection*, in which the goal is to detect similar or identical web pages (e.g., see [13], [18]). Also, we do not cover solutions for problems such as *anaphora resolution* [56], in which the problem is to locate different mentions of the same entity in *free text* (e.g., that the phrase "President of the U.S." refers to the same entity as "George W. Bush"). We should note that the algorithms developed for mirror detection or for anaphora resolution are often applicable for the task of duplicate detection. Techniques for mirror detection have been used for detection of duplicate database records (see, for example, Section V-A.4) and techniques for anaphora resolution are commonly used as an integral part of deduplication in relations that are extracted from free text using information extraction systems [52].

The problem that we study has been known for more than five decades as the *record linkage* or the *record matching* problem [31], [61]–[64], [88] in the statistics community. The goal of record matching is to identify records in the same or different databases that refer to the same real-world entity, even if the records are not identical. In slightly ironic fashion, the same problem has multiple names across research communities. In the database community, the problem is described as *merge-purge* [39], *data deduplication* [78], and *instance identification* [94]; in the AI community, the same problem is described as *database hardening* [21] and *name*

matching [9]. The names *coreference resolution*, *identity uncertainty*, and *duplicate detection* are also commonly used to refer to the same task. We will use the term *duplicate record detection* in this paper.

The remaining part of this paper is organized as follows: In Section II, we briefly discuss the necessary steps in the data cleaning process, *before* the *duplicate record detection* phase. Then, Section III describes techniques used to match individual fields, and Section IV presents techniques for matching records that contain multiple fields. Section V describes methods for improving the efficiency of the duplicate record detection process and Section VI presents a few commercial, off-the-shelf tools used in industry for duplicate record detection and for evaluating the initial quality of the data and of the matched records. Finally, Section VII concludes the paper and discusses interesting directions for future research.

II. Data Preparation

Duplicate record detection is the process of identifying different or multiple records that refer to one unique real-world entity or object. Typically, the process of duplicate detection is preceded by a *data preparation* stage, during which data entries are stored in a uniform manner in the database, resolving (at least partially) the structural heterogeneity problem. The data preparation stage includes a *parsing*, a *data transformation*, and a *standardization* step. The approaches that deal with data preparation are also described under the using the term *ETL* (Extraction, Transformation, Loading) [43]. These steps improve the quality of the in-flow data and make the data comparable and more usable. While data preparation is not the focus of this survey, for completeness we describe briefly the tasks performed in that stage. A comprehensive collection of papers related to various data transformation approaches can be found in [74].

Parsing is the first critical component in the data preparation stage. Parsing locates, identifies and isolates individual data elements in the source files. Parsing makes it easier to correct, standardize, and match data because it allows the comparison of individual components, rather than of long complex strings of data. For example, the appropriate parsing of name and address components into consistent packets of information is a crucial part in the data cleaning process. Multiple parsing methods have been proposed recently in the literature (e.g., [1], [11], [53], [71], [84]) and the area continues to be an active field of research.

Data transformation refers to simple conversions that can be applied to the data in order for

them to conform to the data types of their corresponding domains. In other words, this type of conversion focuses on manipulating one field at a time, without taking into account the values in related fields. The most common form of a simple transformation is the conversion of a data element from one data type to another. Such a data type conversion is usually required when a legacy or parent application stored data in a data type that makes sense within the context of the original application, but not in a newly developed or subsequent system. Renaming of a field from one name to another is considered data transformation as well. Encoded values in operational systems and in external data is another problem that is addressed at this stage. These values should be converted to their decoded equivalents, so records from different sources can be compared in a uniform manner. Range checking is yet another kind of data transformation which involves examining data in a field to ensure that it falls within the expected range, usually a numeric or date range. Lastly, dependency checking is slightly more involved since it requires comparing the value in a particular field to the values in another field, to ensure a minimal level of consistency in the data.

Data standardization refers to the process of standardizing the information represented in certain fields to a specific content format. This is used for information that can be stored in many different ways in various data sources and must be converted to a uniform representation before the duplicate detection process starts. Without standardization, many duplicate entries could erroneously be designated as non-duplicates, based on the fact that common identifying information cannot be compared. One of the most common standardization applications involves address information. There is no one standardized way to capture addresses so the same address can be represented in many different ways. Address standardization locates (using various parsing techniques) components such as house numbers, street names, post office boxes, apartment numbers and rural routes, which are then recorded in the database using a standardized format (e.g., *44 West Fourth Street* is stored as *44 W4th St.*). Date and time formatting and name and title formatting pose other standardization difficulties in a database. Typically, when operational applications are designed and constructed, there is very little uniform handling of date and time formats across applications. Because most operational environments have many different formats for representing dates and times, there is a need to transform dates and times into a standardized format. Name standardization identifies components such as first names, last names, title and middle initials and records everything using some standardized convention. Data standardization

is a rather inexpensive step that can lead to fast identification of duplicates. For example, if the only difference between two records is the differently recorded address (*44 West Fourth Street* vs. *44 W4th St.*), then the data standardization step would make the two records identical, alleviating the need for more expensive approximate matching approaches, that we describe in the later sections.

After the data preparation phase, the data are typically stored in tables, having comparable fields. The next step is to identify which fields should be compared. For example, it would not be meaningful to compare the contents of the field *LastName* with the field *Address*. Perkowitz et al. [67] presented a supervised technique for understanding the "semantics" of the fields that are returned by web databases. The idea was that similar values (e.g. last names) tend to appear in similar fields. Hence, by observing value overlap across fields, it is possible to parse the results into fields and discover correspondences across fields at the same time. Dasu et al. [25] significantly extend this concept and extract a "signature" from each field in the database; this signature summarizes the content of each column in the database. Then, the signatures are used to identify fields with similar values, fields whose contents are subsets of other fields and so on.

Even after parsing, data standardization, and identification of similar fields, it is not trivial to match duplicate records. Misspellings and different conventions for recording the same information still result in different, multiple representations of a unique object in the database. In the next section, we describe techniques for measuring the similarity of individual fields, and later, in Section IV we describe techniques for measuring the similarity of entire records.

III. Field Matching Techniques

One of the most common sources of mismatches in database entries is the typographical variations of string data. Therefore, duplicate detection typically relies on string comparison techniques to deal with typographical variations. Multiple methods have been developed for this task, and each method works well for particular types of errors. While errors might appear in numeric fields as well, the related research is still in its infancy.

In this section, we describe techniques that have been applied for matching fields with string data, in the duplicate record detection context. We also review briefly some common approaches for dealing with errors in numeric data.

A. Character-based similarity metrics

The character-based similarity metrics are designed to handle well typographical errors. In this section, we cover the following similarity metrics:

- Edit distance,
- Affine gap distance,
- Smith-Waterman distance,
- Jaro distance metric, and
- Q-gram distance.

Edit distance: The edit distance between two strings σ_1 and σ_2 is the minimum number of edit operations of single characters needed to transform the string σ_1 into σ_2. There are three types of edit operations:

- *Insert* a character into the string,
- *Delete* a character from the string, and
- *Replace* one character with a different character.

In the simplest form, each edit operation has cost 1. This version of edit distance is also referred to as Levenshtein distance [49]. The basic dynamic programming algorithm [59] for computing the edit distance between two strings takes $O(|\sigma_1| \cdot |\sigma_2|)$ time for two strings of length $|\sigma_1|$ and $|\sigma_2|$, respectively. Landau and Vishkin [48] presented an algorithm for detecting in $O(\max\{|\sigma_1|, |\sigma_2|\} \cdot k)$ whether two strings have edit distance less than k. (Notice that if $||\sigma_1| - |\sigma_2|| > k$ then by definition the two strings do not match within distance k, so $O(\max\{|\sigma_1|, |\sigma_2|\} \cdot k) \sim O(|\sigma_1| \cdot k) \sim O(|\sigma_2| \cdot k)$, for the non-trivial case where $||\sigma_1| - |\sigma_2|| \leq k$.) Needleman and Wunsch [60] modified the original edit distance model, and allowed for different costs for different edit distance operations. (For example, the cost of replacing *O* with *0* might be smaller than the cost of replacing *f* with *q*.) Ristad and Yiannilos [73] presented a method for automatically determining such costs from a set of equivalent words that are written in different ways. The edit distance metrics work well for catching typographical errors, but they are typically ineffective for other types of mismatches.

Affine gap distance: The edit distance metric described above does not work well when matching strings that have been truncated, or shortened (e.g., *"John R. Smith"* vs. *"Jonathan Richard Smith"*). The affine gap distance metric [95] offers a solution to this problem by

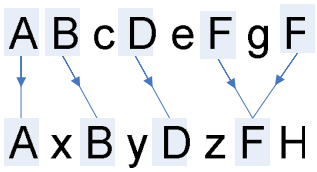

Fig. 1. The Pinheiro and Sun similarity metric alignment for the strings $\sigma_1 = ABcDeFgH$ $\sigma_2 = AxByDzF$.

introducing two extra edit operations: *open gap* and *extend gap*. The cost of extending the gap is usually smaller than the cost of opening a gap, and this results in smaller cost penalties for gap mismatches than the equivalent cost under the edit distance metric. The algorithm for computing the affine gap distance requires $O(a \cdot |\sigma_1| \cdot |\sigma_2|)$ time, when the maximum length of a gap $a \ll \min\{|\sigma_1|, |\sigma_2|\}$. In the general case, the algorithm runs in approximately $O(a^2 \cdot |\sigma_1| \cdot |\sigma_2|)$ steps. Bilenko et al. [9], in a spirit similar to what Ristad and Yiannilos [73] proposed for edit distance, describe how to train an edit distance model with affine gaps.

Smith-Waterman distance: Smith and Waterman [81] described an extension of edit distance and affine gap distance, in which mismatches at the beginning and the end of strings have lower costs than mismatches in the middle. This metric allows for better *local* alignment of the strings (i.e., substring matching). Therefore, the strings *"Prof. John R. Smith, University of Calgary"* and *"John R. Smith, Prof."* can match within short distance using the Smith-Waterman distance, since the prefixes and suffixes are ignored. The distance between two strings can be computed using a dynamic programming technique, based on the Needleman and Wunsch algorithm [60]. The Smith and Waterman algorithm requires $O(|\sigma_1| \cdot |\sigma_2|)$ time and space for two strings of length $|\sigma_1|$ and $|\sigma_2|$; many improvements have been proposed (e.g., the BLAST algorithm [4]), mainly in the context of computational biology applications. Pinheiro and Sun [70] proposed a similar similarity measure, which tries to find the best character alignment for the two compared strings σ_1 and σ_2, so that the number of character mismatches is minimized. For example, the strings $\sigma_1 = ABcDeFgH$ and $\sigma_2 = AxByDzFH$ can be aligned as shown in Figure 1.

Jaro distance metric: Jaro [40] introduced a string comparison algorithm that was mainly used for *comparison of last and first names*. The basic algorithm for computing the Jaro metric for two strings σ_1 and σ_2 includes the following steps:

1) Compute the string lengths $|\sigma_1|$ and $|\sigma_2|$,

2) Find the "common characters" c in the two strings; common are all the characters $\sigma_1[j]$ and $\sigma_2[j]$ for which $\sigma_1[i] = \sigma_2[j]$ and $|i - j| \leq \frac{1}{2}\min\{|\sigma_1|, |\sigma_2|\}$.
3) Find the number of transpositions t; the number of transpositions is computed as follows: We compare the ith common character in σ_1 with the ith common character in σ_2. Each non-matching character is a transposition.

The Jaro comparison value is:

$$\text{Jaro}(\sigma_1, \sigma_2) = \frac{1}{3}\left(\frac{c}{|\sigma_1|} + \frac{c}{|\sigma_2|} + \frac{c - t/2}{c}\right). \quad (1)$$

From the description of the Jaro algorithm, we can see that the Jaro algorithm requires $O(|\sigma_1| \cdot |\sigma_2|)$ time for two strings of length $|\sigma_1|$ and $|\sigma_2|$, mainly due to the Step 2 that computes the "common characters" in the two strings. Winkler and Thibaudeau [101] modified the Jaro metric to give higher weight to prefix matches, since prefix matches are generally more important for surname matching.

Q-grams: The *q-grams* are short character substrings[1] of length q of the database strings [89], [90]. The intuition behind the use of q-grams as a foundation for approximate string matching is that when two strings σ_1 and σ_2 are similar they share a large number of q-grams in common. Given a string σ, its *q-grams* are obtained by "sliding" a window of length q over the characters of σ. Since q-grams at the beginning and the end of the string can have fewer than q characters from σ, the strings are conceptually extended by "padding" the beginning and the end of the string with $q - 1$ occurrences of a special padding character, not in the original alphabet. With the appropriate use of hash-based indexes, the average time required for computing the q-gram overlap between two strings σ_1 and σ_2 is $O(\max\{|\sigma_1|, |\sigma_2|\})$. Letter q-grams, including trigrams, bigrams, and/or unigrams, have been used in a variety of ways in text recognition and spelling correction [47]. One natural extension of q-grams are the *positional q-grams* [83], which also record the position of the q-gram in the string. Gravano et al. [34], [35] showed how to use positional q-grams to locate efficiently similar strings within a relational database.

[1]The q-grams in our context are defined on the character level. In speech processing and in computational linguistics, researchers often use the term *n-gram*, to refer to sequences of n words.

B. Token-based similarity metrics

Character-based similarity metrics work well for typographical errors. However, it is often the case that typographical conventions lead to rearrangement of words (e.g., *"John Smith"* vs. *"Smith, John"*). In such cases, character-level metrics fail to capture the similarity of the entities. Token-based metrics try to compensate for this problem.

Atomic strings: Monge and Elkan [57] proposed a basic algorithm for matching text fields based on *atomic strings*. An *atomic string* is a sequence of alphanumeric characters delimited by punctuation characters. Two atomic strings match if they are equal, or if one is the prefix of the other. Based on this algorithm, the similarity of two fields is the number of their matching *atomic strings* divided by their average number of atomic strings.

WHIRL: Cohen [22] described a system named WHIRL that adopts from the information retrieval the cosine similarity combined with the *tf.idf* weighting scheme to compute the similarity of two fields. Cohen separates each string σ into *words* and each word w is assigned a weight

$$v_\sigma(w) = \log(tf_w + 1) \cdot \log(idf_w)$$

where tf_w is the number of times that w appears in the field and idf_w is $\frac{|D|}{n_w}$, where n_w is the number of records in the database D that contain w. The *tf.idf* weight for a word w in a field is high if w appears a large number of times in the field (large tf_w) and w is a sufficiently "rare" term in the database (large idf_w). For example, for a collection of company names, relatively infrequent terms such as "AT&T" or "IBM" will have higher *idf* weights than more frequent terms such as "Inc." The *cosine similarity* of σ_1 and σ_2 is defined as

$$sim(\sigma_1, \sigma_2) = \frac{\sum_{j=1}^{|D|} v_{\sigma_1}(j) \cdot v_{\sigma_2}(j)}{||v_{\sigma_1}||_2 \cdot ||v_{\sigma_2}||_2}$$

The cosine similarity metric works well for a large variety of entries, and is insensitive to the location of words, thus allowing natural word moves and swaps (e.g., "John Smith" is equivalent to "Smith, John"). Also, introduction of frequent words affects only minimally the similarity of the two strings due to the low *idf* weight of the frequent words. For example, "John Smith" and " Mr. John Smith" would have similarity close to one. Unfortunately, this similarity metric does not capture word spelling errors, especially if they are pervasive and affect many of the words in the strings. For example, the strings "Compter Science Department" and "Deprtment of Computer Scence" will have zero similarity under this metric. Bilenko et al. [9] suggest the

SoftTF-IDF metric to solve this problem. In the SoftTF.IDF metric, pairs of tokens that are "similar"[2] (and not necessarily identical) are also considered in the computation of the cosine similarity. However, the product of the weights for non-identical token pairs is multiplied by the the similarity of the token pair, which is less than one.

Q-grams with tf.idf: Gravano et al. [36] extended the WHIRL system to handle spelling errors by using q-grams, instead of words, as tokens. In this setting, a spelling error minimally affects the set of common q-grams of two strings, so the two strings "Gteway Communications" and "Comunications Gateway" have high similarity under this metric, despite the block move and the spelling errors in both words. This metric handles the insertion and deletion of words nicely. The string "Gateway Communications" matches with high similarity the string "Communications Gateway International" since the q-grams of the word "International" appear often in the relation and have low weight.

C. Phonetic similarity metrics

Character-level and token-based similarity metrics focus on the string-based representation of the database records. However, strings may be phonetically similar even if they are not similar in a character or token level. For example the word *Kageonne* is phonetically similar to *Cajun* despite the fact that the string representations are very different. The phonetic similarity metrics are trying to address such issues and match such strings.

Soundex: Soundex, invented by Russell [75], [76], is the most common *phonetic coding scheme*. Soundex is based on the assignment of identical code digits to phonetically similar groups of consonants and is used mainly to match surnames. The rules of Soundex coding are as follows:

1) Keep the first letter of the surname as the prefix letter and ignore completely all occurrences of W and H in other positions;
2) Assign the following codes to the remaining letters:
   - $B, F, P, V \rightarrow 1$
   - $C, G, J, K, Q, S, X, Z \rightarrow 2$
   - $D, T \rightarrow 3$

[2]The token similarity is measured using a metric that works well for short strings, such as edit distance and Jaro.

- $L \rightarrow 4$
- $M, N \rightarrow 5$
- $R \rightarrow 6$

3) A, E, I, O, U and Y are not coded but serve as separators (see below);
4) Consolidate sequences of identical codes by keeping only the first occurrence of the code;
5) Drop the separators;
6) Keep the letter prefix and the three first codes, padding with zeros if there are fewer than three codes.

Newcombe [61] reports that the Soundex code remains largely unchanged, exposing about two-thirds of the spelling variations observed in linked pairs of vital records, and that it sets aside only a small part of the total discriminating power of the full alphabetic surname. The code is designed primarily for Caucasian surnames, but works well for names of many different origins (such as those appearing on the records of the U.S. Immigration and Naturalization Service). However, when the names are of predominantly East Asian origin, this code is less satisfactory, because much of the discriminating power of these names resides in the vowel sounds, which the code ignores.

New York State Identification and Intelligence System (NYSIIS): The NYSIIS system, proposed by Taft [85], differs from Soundex in that it retains information about the position of vowels in the encoded word by converting most vowels to the letter A. Furthermore, NYSIIS does not use numbers to replace letters; instead it replaces consonants with other, phonetically similar letters, thus returning a purely alpha code (no numeric component). Usually the NYSIIS code for a surname is based on a maximum of nine letters of the full alphabetical name, and the NYSIIS code itself is then limited to six characters. Tafts [85] compared Soundex with NYSIIS, using a name database of New York State, and concluded that NYSIIS is 98.72% accurate, while Soundex is 95.99% accurate for locating surnames. The NYSIIS encoding system is still used today from the New York State Division of Criminal Justice Services.

Oxford Name Compression Algorithm (ONCA): ONCA [33] is a two-stage technique, designed to overcome most of the unsatisfactory features of pure Soundex-ing, retaining in parallel the convenient four-character fixed-length format. In the first step, ONCA uses a British version of the NYSIIS method of compression. Then, in the second step, the transformed and partially

compressed name is Soundex-ed in the usual way. This two-stage technique has been used successfully for grouping similar names together.

Metaphone and Double Metaphone: Philips [68] suggested the *Metaphone* algorithm as a better alternative to Soundex. Philips suggested using 16 consonant sounds that can describe a large number of sounds used in many English and non-English words. *Double Metaphone* [69] is a better version of *Metaphone*, improving some encoding choices made in the initial Metaphone and allowing multiple encodings for names that have various possible pronunciations. For such cases, all possible encodings are tested when trying to retrieve similar names. The introduction of multiple phonetic encodings greatly enhances the matching performance, with rather small overhead. Philips suggested that, at most, 10% of American surnames have multiple encodings.

D. Numeric Similarity Metrics

While multiple methods exist for detecting similarities of string-based data, the methods for capturing similarities in numeric data are rather primitive. Typically, the numbers are treated as strings (and compared using the metrics described above) or simple range queries, which locate numbers with similar values. Koudas et al. [46] suggest, as direction for future research, consideration of the distribution and type of the numeric data, or extending the notion of cosine similarity for numeric data [2] to work well for duplicate detection purposes.

E. Concluding Remarks

The large number of field comparison metrics reflects the large number of errors or transformations that may occur in real-life data. Unfortunately, there are very few studies that compare the effectiveness of the various distance metrics presented here. Yancey [103] shows that the Jaro-Winkler metric works well for name matching tasks for data coming from U.S. census. A notable comparison effort is the work of Bilenko et al. [9], who compare the effectiveness of character-based and token-based similarity metrics. They show that the *Monge-Elkan* metric has the highest average performance across data sets and across character-based distance metrics. They also show that the *SoftTF.IDF* metric works better than any other metric. However, Bilenko et al. emphasize that no single metric is suitable for all data sets. Even metrics that demonstrate robust and high performance for some data sets can perform poorly on others. Hence, they

advocate more flexible metrics that can accommodate multiple similarity comparisons (e.g., [9], [87]). In the next section we review such approaches.

IV. Detecting Duplicate Records

In the previous section we described methods that can be used to match individual fields of a record. In most real-life situations, however, the records consist of multiple fields, making the duplicate detection problem much more complicated. In this section, we review methods that are used for matching records with multiple fields. The presented methods can be broadly divided into two categories:

- Approaches that rely on training data to "learn" how to match the records. This category includes (some) probabilistic approaches and supervised machine learning techniques.
- Approaches that rely on domain knowledge or on generic distance metrics to match records. This category includes approaches that use declarative languages for matching, and approaches that devise distance metrics appropriate for the duplicate detection task.

The rest of this section is organized as follows: initially, in Section IV-A we describe the notation. In Section IV-B we present probabilistic approaches for solving the duplicate detection problem. In Section IV-C we list approaches that use supervised machine learning techniques and in Section IV-D we describe variations based on active learning methods. Section IV-E describes distance-based methods and Section IV-F describes declarative techniques for duplicate detection. Finally, Section IV-G covers unsupervised machine learning techniques, and Section IV-H provides some concluding remarks.

A. Notation

We use A and B to denote the tables that we want to match, and we assume, without loss of generality, that A and B have n comparable fields. In the duplicate detection problem, each tuple pair $\langle \alpha, \beta \rangle$, $(\alpha \in A, \beta \in B)$, is assigned to one of the two classes M and U. The class M contains the record pairs that represent the same entity ("*match*") and the class U contains the record pairs that represent two different entities ("*non-match*").

We represent each tuple pair $\langle \alpha, \beta \rangle$ as a random vector $\underline{x} = [x_1, \ldots, x_n]^T$ with n components that correspond to the n comparable fields of A and B. Each x_i shows the level of agreement

of the ith field for the records α and β. Many approaches use binary values for the x_i's and set $x_i = 1$ if field i agrees and let $x_i = 0$ if field i disagrees.

B. Probabilistic Matching Models

Newcombe et al. [64] were the first to recognize duplicate detection as a Bayesian inference problem. Then, Fellegi and Sunter [31] formalized the intuition of Newcombe et al. and introduced the notation that we use, which is also commonly used in duplicate detection literature. The comparison vector $\underline{x}$ is the input to a decision rule that assigns $\underline{x}$ to U or to M. The main assumption is that $\underline{x}$ is a random vector whose density function is different for each of the two classes. Then, if the density function for each class is known, the duplicate detection problem becomes a Bayesian inference problem. In the following sections, we will discuss various techniques that have been developed for addressing this (general) decision problem.

1) The Bayes Decision Rule for Minimum Error: Let $\underline{x}$ be a comparison vector, randomly drawn from the comparison space that corresponds to the record pair $\langle \alpha, \beta \rangle$. The goal is to determine whether $\langle \alpha, \beta \rangle \in M$ or $\langle \alpha, \beta \rangle \in U$. A decision rule, based simply on probabilities, can be written as follows:

$$\langle \alpha, \beta \rangle \in \begin{cases} M & \text{if } p(M|\underline{x}) \geq p(U|\underline{x}) \\ U & \text{otherwise} \end{cases} \tag{2}$$

This decision rule indicates that if the probability of the match class M, given the comparison vector $\underline{x}$, is larger than the probability of the non-match class U, then $\underline{x}$ is classified to M, and vice versa. By using the Bayes theorem, the previous decision rule may be expressed as:

$$\langle \alpha, \beta \rangle \in \begin{cases} M & \text{if } l(\underline{x}) = \frac{p(\underline{x}|M)}{p(\underline{x}|U}) \geq \frac{p(U)}{p(M)} \\ U & \text{otherwise} \end{cases} \tag{3}$$

The ratio

$$l(\underline{x}) = \frac{p(\underline{x}|M)}{p(\underline{x}|U)} \tag{4}$$

is called the *likelihood ratio*. The ratio $\frac{p(U)}{p(M)}$ denotes the threshold value of the likelihood ratio for the decision. We refer to the decision rule in Equation 3 as the *Bayes test* for *minimum error*. It can be easily shown [38] that the Bayes test results in the smallest probability of error, and it is in that respect an optimal classifier. Of course this holds only when the distributions

of $p(\underline{x}|M)$, $p(\underline{x}|U)$ and the priors $p(U)$ and $p(M)$ are known; this unfortunately is very rarely the case.

One common approach, usually called *Naive Bayes*, to compute the distributions of $p(\underline{x}|M)$ and $p(\underline{x}|U)$ is to make a conditional independence assumption, and postulate that the probabilities $p(x_i|M)$ and $p(x_j|M)$ are independent if $i \neq j$. (Similarly, for $p(x_i|U)$ and $p(x_j|U)$.) In that case, we have

$$\begin{aligned} p(\underline{x}|M) &= \prod_{i=i}^{n} p(x_i|M) \\ p(\underline{x}|U) &= \prod_{i=i}^{n} p(x_i|U) \end{aligned}$$

The values of $p(x_i|M)$ and $p(x_i|U)$ can be computed using a training set of pre-labeled record pairs. However, the probabilistic model can also be used without using training data. Jaro [41] used a binary model for the values of x_i (i.e., if the field i "matches" $x_i = 1$, else $x_i = 0$) and suggested using an expectation maximization (EM) algorithm [26] to compute the probabilities $p(x_i = 1|M)$. The probabilities $p(x_i = 1|U)$ can be estimated by taking random pairs of records (which are with high probability in U).

When the conditional independence is not a reasonable assumption, then Winkler [97] suggested using the *general expectation maximization* algorithm to estimate $p(\underline{x}|M)$, $p(\underline{x}|U)$. In [99], Winkler claims that the general, unsupervised EM algorithm works well under five conditions:

1) the data contain a relatively large percentage of matches (more than 5%),
2) the matching pairs are "well-separated" from the other classes,
3) the rate of typographical errors is low,
4) there are sufficiently many redundant identifiers to overcome errors in other fields of the record, and
5) the estimates computed under the conditional independence assumption result in good classification performance.

Winkler [99] shows how to relax the assumptions above (including the conditional independence assumption) and still get good matching results. Winkler shows that a semi-supervised model, which combines labeled and unlabeled data (similar to Nigam et al. [65]), performs better than purely unsupervised approaches. When no training data is available, unsupervised EM works well, even when a limited number of interactions is allowed between the variables. Interestingly,

the results under the independence assumption are not considerably worse compared to the case in which the EM model allows variable interactions.

Du Bois [28] pointed out the importance of the fact that many times fields have missing (null) values and proposed a different method to correct mismatches that occur due to missing values. Du Bois suggested using a new comparison vector $\underline{x}^*$ with dimension $2n$ instead of the n-dimensional comparison vector $\underline{x}$, such that

$$\underline{x}^* = (x_1, x_2, \ldots, x_n, x_1y_1, x_2y_2, \ldots, x_ny_n) \tag{5}$$

where

$$y_i = \begin{cases} 1 & \text{if the } i\text{-th field on both records is present,} \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Using this representation, mismatches that occur due to missing data are typically discounted, resulting in improved duplicate detection performance. Du Bois proposed using an independence model to learn the distributions of $p(x_iy_i|M)$ and $p(x_iy_i|U)$ by using a set of pre-labeled training record pairs.

2) The Bayes Decision Rule for Minimum Cost: Often, in practice, the minimization of the probability of error is not the best criterion for creating decision rules, as the misclassifications of M and U samples may have different consequences. Therefore, it is appropriate to assign a cost c_{ij} to each situation, which is the cost of deciding that $\underline{x}$ belongs to the class i when $\underline{x}$ actually belongs to the class j. Then, the expected costs $r_M(\underline{x})$ and $r_U(\underline{x})$ of deciding that $\underline{x}$ belongs to the class M and U, respectively, are:

$$\begin{aligned} r_M(\underline{x}) &= c_{MM} \cdot p(M|\underline{x}) + c_{MU} \cdot p(U|\underline{x}) \\ r_U(\underline{x}) &= c_{UM} \cdot p(M|\underline{x}) + c_{UU} \cdot p(U|\underline{x}) \end{aligned}$$

In that case, the decision rule for assigning $\underline{x}$ to M becomes:

$$\langle \alpha, \beta \rangle \in \begin{cases} M & \text{if } r_M(\underline{x}) < r_U(\underline{x}) \\ U & \text{otherwise} \end{cases} \tag{7}$$

It can be easily proved [29] that the minimum cost decision rule for the problem can be stated as:

$$\langle \alpha, \beta \rangle \in \begin{cases} M & \text{if } l(\underline{x}) > \frac{(c_{MU} - c_{UU}) \cdot p(U)}{(c_{UM} - c_{MM}) \cdot p(M)} \\ U & \text{otherwise} \end{cases} \tag{8}$$

Comparing the minimum error and minimum cost decision rule, we notice that the two decision rules become the same for the special setting of the cost functions to $c_{UM} - c_{MM} = c_{MU} - c_{UU}$. In this case, the cost functions are termed symmetrical. For a symmetrical cost function, the cost becomes the probability of error and the Bayes test for minimum cost specifically addresses and minimizes this error.

3) Decision with a Reject Region: Using the Bayes Decision rule when the distribution parameters are known leads to optimal results. However, even in an ideal scenario, when the likelihood ratio $l(\underline{x})$ is close to the threshold, the error (or cost) of *any* decision is high [29]. Based on this well-known and general idea in decision theory, Fellegi and Sunter [31], suggested adding an extra "*reject*" class in addition to the classes M and U. The *reject* class contained record pairs for which it is not possible to make any definite inference, and a "clerical review" is necessary. These pairs are examined manually by experts to decide whether they are true matches or not. By setting thresholds for the conditional error on M and U, we can define the *reject region* and the *reject probability*, which measure the probability of directing a record pair to an expert for review.

Tepping [88] was the first to suggest a solution methodology focusing on the costs of the decision. He presented a graphical approach for estimating the likelihood thresholds. Verykios et al. [93] developed a formal framework for the cost-based approach taken by Tepping which shows how to compute the thresholds for the three decision areas when the costs and the priors $P(M)$ and $P(U)$ are known.

The "reject region" approach can be easily extended to a larger number of decision areas [92]. The main problem with such a generalization is appropriately ordering the thresholds which determine the regions in a way that no region disappears.

C. Supervised and Semi-Supervised Learning

The probabilistic model uses a Bayesian approach to classify record pairs into two classes, M and U. This model was widely used for duplicate detection tasks, usually as an application of the Fellegi-Sunter model. While the Fellegi-Sunter approach dominated the field for more than two decades, the development of new classification techniques in the machine learning and statistics communities prompted the development of new deduplication techniques. The

supervised learning systems rely on the existence of training data in the form of record pairs, pre-labeled as matching or not.

One set of supervised learning techniques treat each record pair $\langle \alpha, \beta \rangle$ independently, similarly to the probabilistic techniques of Section IV-B. Cochinwala et al. [19] used the well-known CART algorithm [12], which generates classification and regression trees, a linear discriminant algorithm [38], which generates linear combination of the parameters for separating the data according to their classes, and a "vector quantization" approach, which is a generalization of nearest neighbor algorithms. The experiments which were conducted indicate that CART has the smallest error percentage. Bilenko et al. [9] use SVMlight [42] to learn how to merge the matching results for the individual fields of the records. Bilenko et al. showed that the SVM approach usually outperforms simpler approaches, such as treating the whole record as one large field. A typical post-processing step for these techniques (including the probabilistic techniques of Section IV-B) is to construct a graph for all the records in the database, linking together the matching records. Then, using the transitivity assumption, all the records that belong to the same connected component are considered identical [58].

The transitivity assumption can sometimes result in inconsistent decisions. For example, $\langle \alpha, \beta \rangle$ and $\langle \alpha, \gamma \rangle$ can be considered matches, but $\langle \beta, \gamma \rangle$ not. Partitioning such "inconsistent" graphs with the goal of minimizing inconsistencies is an NP-complete problem [6]. Bansal et al. [6] propose a polynomial approximation algorithm that can partition such a graph, identifying automatically the clusters and the number of clusters in the dataset. Cohen and Richman [23] proposed a supervised approach in which the system learns from training data how to cluster together records that refer to the same real-world entry. The main contribution of this approach is the adaptive distance function which is learned from a given set of training examples. McCallum and Wellner [55] learn the clustering method using training data; their technique is equivalent to a graph partitioning technique that tries to find the min-cut *and* the appropriate number of clusters for the given data set, similarly to the work of Bansal et al. [6].

The supervised clustering techniques described above have *records* as nodes for the graph. Singla and Domingos [80] observed that by using attribute values as nodes, it is possible to propagate information across nodes and improve duplicate record detection. For example, if the records $\langle Google, MountainView, CA \rangle$ and $\langle GoogleInc., MountainView, California \rangle$ are deemed equal, then CA and $California$ are also equal, and this information can be useful

for other record comparisons. The underlying assumption is that the only differences are due to different representations of the same entity (e.g., "Google" and "Google Inc.") and that there is no erroneous information in the attribute values (e.g., by mistake someone entering $Bismarck, ND$ as the location of Google headquarters). Pasula et al. [66] propose a semi-supervised probabilistic relational model that can handle a generic set of transformations. While the model can handle a large number of duplicate detection problems, the use of exact inference results in a computationally intractable model. Pasula et al. propose to use a Markov Chain Monte Carlo (MCMC) sampling algorithm to avoid the intractability issue. However, it is unclear whether techniques that rely on graph-based probabilistic inference can scale well for data sets with hundreds of thousands of records.

D. Active-Learning-Based Techniques

One of the problems with the supervised learning techniques is the requirement for a large number of training examples. While it is easy to create a large number of training pairs that are either clearly non-duplicates or clearly duplicates, it is very difficult to generate ambiguous cases that would help create a highly accurate classifier. Based on this observation, some duplicate detection systems used active learning techniques [24] to automatically locate such ambiguous pairs. Unlike an "ordinary" learner that is trained using a static training set, an "active" learner actively picks subsets of instances from unlabeled data, which, when labeled, will provide the highest information gain to the learner.

Sarawagi and Bhamidipaty [78] designed ALIAS, a learning based duplicate detection system, that uses the idea of a "reject region" (see Section IV-B.3) to significantly reduce the size of the training set. The main idea behind ALIAS is that most duplicate and non-duplicate pairs are clearly distinct. For such pairs, the system can automatically categorize them in U and M without the need of manual labeling. ALIAS requires humans to label pairs only for cases where the uncertainty is high. This is similar to the "reject region" in the Fellegi and Sunter model, which marked ambiguous cases as cases for clerical review.

ALIAS starts with small subsets of pairs of records designed for training, which have been characterized as either matched or unique. This initial set of labeled data forms the training data for a preliminary classifier. In the sequel, the initial classifier is used for predicting the status of unlabeled pairs of records. The initial classifier will make clear determinations on some unlabeled

instances but lack determination on most. The goal is to seek out from the unlabeled data pool those instances which, when labeled, will improve the accuracy of the classifier at the fastest possible rate. Pairs whose status is difficult to determine serve to strengthen the integrity of the learner. Conversely, instances in which the learner can easily predict the status of the pairs do not have much effect on the learner. Using this technique, ALIAS can quickly learn the peculiarities of a data set and rapidly detect duplicates using only a small number of training data.

Tejada et al. [86], [87] used a similar strategy and employed decision trees to teach rules for matching records with multiple fields. Their method suggested that by creating multiple classifiers, trained using slightly different data or parameters, it is possible to detect ambiguous cases and then ask the user for feedback. The key innovation in this work is the creation of several redundant functions and the concurrent exploitation of their conflicting actions in order to discover new kinds of inconsistencies among duplicates in the data set.

E. Distance-Based Techniques

Even active learning techniques require some training data or some human effort to create the matching models. In the absence of such training data or ability to get human input, supervised and active learning techniques are not appropriate. One way of avoiding the need for training data is to define a distance metric for *records*, which does not need tuning through training data. Using the distance metric and an appropriate matching threshold, it is possible to match similar records, without the need for training.

One approach is to treat a record as a long field, and use one of the distance metrics described in Section III to determine which records are similar. Monge and Elkan [57], [58] proposed a string matching algorithm for detecting highly similar database records. The basic idea was to apply a general purpose field matching algorithm, especially one that is able to account for gaps in the strings, to play the role of the duplicate detection algorithm. Similarly, Cohen [20] suggested to use the *tf.idf* weighting scheme (see Section III-B), together with the cosine similarity metric to measure the similarity of records. Koudas et al. [46] presented some practical solutions to problems encountered during the deployment of such a string-based duplicate detection system at AT&T.

Distance-based approaches that conflate each record in one big field may ignore important information that can be used for duplicate detection. A simple approach is to measure the distance

between individual fields, using the appropriate distance metric for each field, and then compute the weighted distance [27] between the records. In this case, the problem is the computation of the weights, and the overall setting becomes very similar to the probabilistic setting that we discussed in Section IV-B. An alternative approach, proposed by Guha et al. [37] is to create a distance metric that is based on ranked list merging. The basic idea is that if we compare only one field from the record, the matching algorithm can easily find the best matches and rank them according to their similarity, putting the best matches first. By applying the same principle for all the fields, we can get, for each record, n ranked lists of records, one for each field. Then, the goal is to create a rank of records that has the minimum aggregate *rank distance* when compared to all the n lists. Guha et al. map the problem into the minimum cost perfect matching problem, and develop then efficient solutions for identifying the top-k matching records. The first solution is based on the *Hungarian Algorithm* [3], a graph-theoretic algorithm that solves the minimum cost perfect matching problem. Guha et al. also present the *Successive Shortest Paths* algorithm that works well for smaller values of k and is based on the idea that it is not required to examine all potential matches to identify the top-k matches. Both of the proposed algorithms are implemented in T-SQL and are directly deployable over existing relational databases.

The distance-based techniques described so far, treat each record as a flat entity, ignoring the fact that data is often stored in relational databases, in multiple tables. Ananthakrishna et al. [5] describe a similarity metric that uses not only the textual similarity, but the "co-occurrence" similarity of two entries in a database. For example, the entries in the state column "CA" and "California" have small textual similarity; however, the city entries "San Francisco," "Los Angeles," "San Diego" and so on, often have foreign keys that point both to "CA" and "California." Therefore, it is possible to infer that "CA" and "California" are equivalent. Ananthakrishna et al. show that by using "foreign key co-occurrence" information, they can substantially improve the quality of duplicate detection in databases that use multiple tables to store the entries of a record. This approach is conceptually similar to the work of Perkowitz et al. [67] and of Dasu et al. [25], which examine the contents of *fields* to locate the matching fields across two tables (see Section II).

Finally, one of the problems of the distance-based techniques is the need to define the appropriate value for the matching threshold. In the presence of training data, it is possible to find the appropriate threshold value. However, this would nullify the major advantage of distance-

based techniques, which is the ability to operate without training data. Recently, Chaudhuri et al. [16] proposed a new framework for distance-based duplicate detection, observing that the distance thresholds for detecting real duplicate entries is different from each database tuple. To detect the appropriate threshold, Chaudhuri et al. observed that entries that correspond to the same real-world object but have different representation in the database, tend to (1) have small distances from each other (compact set property), and to (2) have only a small number of other neighbors within a small distance (sparse neighborhood property). Furthermore, Chaudhuri et al. propose an efficient algorithm for computing the required threshold for each object in the database, and show that the quality of the results outperforms approaches that rely on a single, global threshold.

F. Rule-based Approaches

A special case of distance-based approaches is the use of rules to define whether two records are the same or not. Rule-based approaches can be considered as distance-based techniques, where the distance of two records is either 0 or 1. Wang and Madnick [94] proposed a rule-based approach for the duplicate detection problem. For cases in which there is no global key, Wang and Madnick suggest the use of rules developed by experts to derive a set of attributes that collectively serve as a "key" for each record. For example, an expert might define rules such as

IF age<22	THEN status = undergraduate ELSE status = graduate
IF distanceFromHome >10	THEN transportation = car ELSE transportation = bicycle

By using such rules, Wang and Madnick hoped to generate unique keys that can cluster multiple records that represent the same real-world entity. Lim et al. [50] also used a rule-based approach, but with the extra restriction that the result of the rules must always be correct. Therefore, the rules should not be heuristically-defined but should reflect absolute truths and serve as functional dependencies.

Hernández and Stolfo [39] further developed this idea and derived an equational theory that dictates the logic of domain equivalence. This equational theory specifies an inference about the

similarity of the records. For example, if two persons have similar name spellings, and these persons have the same address, we may infer that they are the same person. Specifying such an inference in the equational theory requires declarative rule language. For example, the following is a rule that exemplifies one axiom of the equational theory developed for an employee database:

```
FORALL (r1,r2) in EMPLOYEE
  IF r1.name is similar to r2.name AND
     r1.address = r2.address
  THEN r1 matches r2
```

Note that "similar to" is measured by one of the string comparison techniques (Section III), and "matches" means to declare that those two records are matched and therefore represent the same person.

AJAX [32] is a prototype system that provides a declarative language for specifying data cleaning programs, consisting of SQL statements enhanced with a set of primitive operations to express various cleaning transformations. AJAX provides a framework wherein the logic of a data cleaning program is modeled as a directed graph of data transformations starting from some input source data. Four types of data transformations are provided to the user of the system. The mapping transformation standardizes data, the matching transformation finds pairs of records that probably refer to the same real object, the clustering transformation groups together matching pairs with a high similarity value, and finally, the merging transformation collapses each individual cluster into a tuple of the resulting data source.

It is noteworthy that such rule-based approaches, which require a human expert to devise meticulously crafted matching rules, typically result in systems with high accuracy. However, the required tuning requires extremely high manual effort from the human experts, and this effort makes the deployment of such systems difficult in practice. Currently, the typical approach is to use a system that generates matching rules from training data (see Sections IV-C and IV-D) and then manually tune the automatically generated rules.

G. Unsupervised Learning

As we mentioned earlier, the comparison space consists of comparison vectors which contain information about the differences between fields in a pair of records. Unless some information

exists about which comparison vectors correspond to which category (match, non-match, or possible-match), the labeling of the comparison vectors in the training data set should be done manually. One way to avoid manual labeling of the comparison vectors is to use clustering algorithms, and group together similar comparison vectors. The idea behind most unsupervised learning approaches for duplicate detection is that similar comparison vectors correspond to the same class.

The idea of unsupervised learning for duplicate detection has its roots in the probabilistic model proposed by Fellegi and Sunter (see Section IV-B). As we discussed in Section IV-B, when there are no training data to compute the probability estimates, it is possible to use variations of the Expectation Maximization algorithm to identify appropriate clusters in the data.

Verykios et al. [91] propose the use of a bootstrapping technique based on clustering to learn matching models. The basic idea, also known as co-training [10], is to use very few labeled data, and then use unsupervised learning techniques to label appropriately the data with unknown labels. Initially, Verykios et al. treat each entry of the comparison vector (which corresponds to the result of a field comparison) as a continuous, real variable. Then, they partition the comparison space into clusters by using the AutoClass [17] clustering tool. The basic premise is that each cluster contains comparison vectors with similar characteristics. Therefore all the record pairs in the cluster belong to the same class (matches, non-matches, or possible-matches). Thus, by knowing the real class of only a few vectors in each cluster, it is possible to infer the class of all vectors in the cluster, and therefore mark the corresponding record pairs as matches or not. Elfeky et al. [30] implemented this idea in TAILOR, a toolbox for detecting duplicate entries in data sets. Verykios et al. show that the classifiers generated using the new, larger training set have high accuracy, and require only a minimal number of pre-labeled record pairs.

Ravikumar and Cohen [72] follow a similar approach and propose a hierarchical, graphical model for learning to match record pairs. The foundation of this approach is to model each field of the comparison vector as a latent binary variable which shows whether the two fields match or not. The latent variable then defines two probability distributions for the values of the corresponding "observed" comparison variable. Ravikumar and Cohen show that it is easier to learn the parameters of a hierarchical model than to attempt to directly model the distributions of the real-valued comparison vectors. Bhattacharya and Getoor [8] propose to use the Latent Dirichlet Allocation generative model to perform duplicate detection. In this model, the latent

variable is a unique identifier for each entity in the database.

H. Concluding Remarks

There are multiple techniques for duplicate record detection. We can divide the techniques into two broad categories: ad-hoc techniques that work quickly on existing relational databases, and more "principled" techniques that are based on probabilistic inference models. While probabilistic methods outperform ad-hoc techniques in terms of accuracy, the ad-hoc techniques work much faster and can scale to databases with hundreds of thousands of records. Probabilistic inference techniques are practical today only for data sets that are one or two orders of magnitude smaller than the data sets handled by ad-hoc techniques. A promising direction for future research is to devise techniques that can substantially improve the efficiency of approaches that rely on machine learning and probabilistic inference.

A question that is unlikely to be resolved soon is the question of which of the presented methods should be used for a given duplicate detection task. Unfortunately, there is no clear answer to this question. The duplicate record detection task is highly data-dependent and it is unclear if we will ever see a technique dominating all others across all data sets. The problem of choosing the best method for duplicate data detection is very similar to the problem of *model selection* and *performance prediction* for data mining: we expect that progress in that front will also benefit the task of selecting the best method for duplicate detection.

V. Improving the Efficiency of Duplicate Detection

So far, in our discussion of methods for detecting whether two records refer to the same real-world object, we have focused mainly on the *quality* of the comparison techniques and not on the efficiency of the duplicate detection process. Now, we turn to the central issue of improving the speed of duplicate detection.

An elementary technique for discovering matching entries in tables A and B is to execute a "nested-loop" comparison, i.e., to compare every record of table A with every record in table B. Unfortunately, such strategy requires a total of $|A| \cdot |B|$ comparisons, a cost that is prohibitively expensive even for the moderately-sized tables. In Section V-A we describe techniques that substantially reduce the number of required comparisons.

Another factor that can lead to increased computation expense is the cost required for a *single* comparison. It is not uncommon for a record to contain tens of fields. Therefore, each record comparison requires multiple field comparisons and each field comparison can be expensive. For example, computing the edit distance between two long strings σ_1 and σ_2, respectively, has a cost of $O(|\sigma_1| \cdot |\sigma_2|)$; just checking if they are within a prespecified edit distance threshold k can reduce the complexity to $O(\max\{|\sigma_1|, |\sigma_2|\} \cdot k)$ (see Section III-A). We examine some of the methods that can be used to reduce the cost of record comparison in Section V-B.

A. Reducing the Number of Record Comparisons

1) Blocking: One "traditional" method for identifying identical records in a database table is to scan the table and compute the value of a hash function for each record. The value of the hash function defines the "bucket" to which this record is assigned. By definition, two records that are identical will be assigned to the same bucket. Therefore, in order to locate duplicates, it is enough to compare only the records that fall into the same bucket for matches. The hashing technique cannot be used directly for approximate duplicates, since there is no guarantee that the hash value of two similar records will be the same. However, there is an interesting counterpart of this method, named *blocking*.

As discussed above with relation to utilizing the hash function, *blocking* typically refers to the procedure of subdividing files into a set of mutually exclusive subsets (blocks) under the assumption that no matches occur across different blocks. A common approach to achieving these blocks is to use a function such as Soundex, NYSIIS, or Metaphone (see Section III-C) on highly discriminating fields (e.g., last name) and compare only records that have similar, but not necessarily identical, fields.

Although blocking can increase substantially the speed of the comparison process, it can also lead to an increased number of false mismatches due to the failure of comparing records that do not agree on the blocking field. It can also lead to an increased number of missed matches due to errors in the blocking step that placed entries in the wrong buckets, thereby preventing them from being compared to actual matching entries. One alternative is to execute the duplicate detection algorithm in multiple runs, each time using a different field for blocking. This approach can reduce substantially the probability of false mismatches, with a relatively small increase in the running time.

2) Sorted Neighborhood Approach: Hernández and Stolfo [39] describe the so-called *sorted neighborhood* approach. The method consists of the following three steps:

- *Create key:* A key for each record in the list is computed by extracting relevant fields or portions of fields.
- *Sort data:* The records in the database are sorted by using the key found in the first step. A sorting key is defined to be a sequence of attributes, or a sequence of sub-strings within the attributes, chosen from the record in an ad hoc manner. Attributes that appear first in the key have a higher priority than those that appear subsequently.
- *Merge:* A fixed size window is moved through the sequential list of records in order to limit the comparisons for matching records to those records in the window. If the size of the window is w records then every new record that enters that window is compared with the previous $w-1$ records to find "matching" records. The first record in the window slides out of it.

The *sorted neighborhood* approach relies on the assumption that duplicate records will be close in the sorted list, and therefore will be compared during the merge step. The effectiveness of the sorted neighborhood approach is highly dependent upon the comparison key that is selected to sort the records. In general, no single key will be sufficient to sort the records in such a way that all the matching records can be detected. If the error in a record occurs in the particular field or portion of the field that is the most important part of the sorting key, there is a very small possibility that the record will end up close to a matching record after sorting.

To increase the number of similar records merged, Hernández and Stolfo implemented a strategy for executing several independent runs of the sorted-neighborhood method (presented above) by using a different sorting key and a relatively small window each time. This strategy is called the *multi-pass* approach. This method is similar in spirit to the multiple-run blocking approach described above. Each independent run produces a set of pairs of records that can be merged. The final results, including the transitive closure of the records matched in different passes, is subsequently computed.

3) Clustering and Canopies: Monge and Elkan [58] try to improve the performance of a basic "nested-loop" record comparison, by assuming that duplicate detection is transitive. This means that if α is deemed duplicate of β and β is deemed duplicate of γ then α and γ are also duplicates. Under the assumption of transitivity, the problem of matching records in a database

can be described in terms of determining the connected components of an undirected graph. At any time, the connected components of the graph correspond to the transitive closure of the "record matches" relationships discovered so far. Monge and Elkan [58] use a *union-find* structure to efficiently compute the connected components of the graph. During the *Union* step, duplicate records are "merged" into a cluster and only a "representative" of the cluster is kept for subsequent comparisons. This reduces the total number of record comparisons, without reducing substantially the accuracy of the duplicate detection process. The concept behind this approach, is that if a record α is not similar to a record β already in the cluster, then it will not match the other members of the cluster either.

McCallum et al. [54] propose the use of *canopies* for speeding up the duplicate detection process. The basic idea is to use a cheap comparison metric to group records into *overlapping* clusters called canopies. (This is in contrast to blocking that requires hard, non-overlapping partitions.) After the first step, the records are then compared pairwise, using a more expensive similarity metric that leads to better qualitative results. The assumption behind this method is that there is an inexpensive similarity function that can be used as a "quick-and-dirty" approximation for another, more expensive function. For example, if two strings have length difference larger than 3, then their edit distance cannot be smaller than 3. In that case, the length comparison serves as a cheap (canopy) function for the more expensive edit distance. Cohen and Richman [23] propose the tf.idf similarity metric as a canopy distance, and then use multiple (expensive) similarity metrics to infer whether two records are duplicates. Gravano et al. [35] propose using the string lengths and the number of common q-grams of two strings as canopies (filters according to [35]) for the edit distance metric, which is expensive to compute in a relational database. The advantage of this technique is that the canopy functions can be evaluated efficiently using vanilla SQL statements. In a similar fashion, Chaudhuri et al. [15] propose using an *indexable* canopy function for easily identifying similar tuples in a database. Baxter et al. [7] perform an experimental comparison of canopy-based approaches with traditional blocking and show that the flexible nature of canopies can significantly improve the quality and speed of duplicate detection.

4) Set Joins: Another direction towards efficiently implementing data cleaning operations is to speed-up the execution of set operations: large number of similarity metrics, discussed in Section III, use set operations as part of the overall computation. Running set operations on all

pair combinations is a computationally expensive operation and is typically unnecessary. For data cleaning applications, the interesting pairs are only those in which the similarity value is high. Many techniques use this property and suggest algorithms for fast computation of set-based operations on a set of records.

Cohen [20] proposed using a set of in-memory inverted indexes together with an $A*$ search algorithm to locate the top-k most similar pairs, according to the cosine similarity metric. Soffer et al. [82], mainly in the context of information retrieval, suggest pruning the inverted index, removing terms with low weights since they do not contribute much to the computation of the *tf.idf* cosine similarity. Gravano et al. [36] present an SQL-based approach that is analogous to the approach of Soffer et al. [82], and allows fast computation of cosine similarity within an RDBMS. Mamoulis [51] presents techniques for efficiently processing a set join in a database, focusing on the containment and non-zero-overlap operators. Mamoulis shows that inverted indexes are typically superior to approaches based on signature files, confirming earlier comparison studies [104]. Sarawagi and Kirpal [79] extend the set joins approach to a large number of similarity predicates that use set joins. The *Probe-Cluster* approach of Sarawagi and Kirpal works well in environments with limited main memory, and can be used to compute efficiently a large number of similarity predicates, in contrast to previous approaches which were tuned for a smaller number of similarity predicates (e.g., set containment, or cosine similarity). Furthermore, *Probe-Cluster* returns exact values for the similarity metrics, in contrast to previous approaches which used approximation techniques.

B. Improving the Efficiency of Record Comparison

So far, we have examined techniques that reduce the number of required record comparisons without compromising the quality of the duplicate detection process. Another way of improving the efficiency of duplicate detection is to improve the efficiency of a *single* record comparison. Next, we review some of these techniques.

When comparing two records, after having computed the differences of only a small portion of the fields of two records, it may be obvious that the pair does match, irrespective of the results of further comparison. Therefore, it is paramount to determine the field comparison for a pair of records as soon as possible to avoid wasting additional, valuable time. The field comparisons should be terminated when even complete agreement of all the remaining fields cannot reverse

the unfavorable evidence for the matching of the records [62]. To make the early termination work, the global likelihood ratio for the full agreement of each of the identifiers should be calculated. At any given point in the comparison sequence, the maximum collective favorable evidence, which could be accumulated from that point forward, will indicate what improvement in the overall likelihood ratio might conceivably result if the comparisons were continued.

Verykios et al. [91] propose a set of techniques for reducing the complexity of record comparison. The first step is to apply a feature subset selection algorithm for reducing the dimensionality of the input set. By using a feature selection algorithm (e.g., [44]) as a preprocessing step the record comparison process uses only a small subset of the record fields, which speeds up the comparison process. Additionally, the induced model can be generated in a reduced amount of time and is usually characterized by higher predictive accuracy. Verykios et al. [91] also suggest using a pruning technique on the derived decision trees that are used to classify record pairs as matches or mismatches. Pruning produces models (trees) of smaller size not only avoid overfitting and have a higher accuracy, but also allow for faster execution of the matching algorithm.

VI. Duplicate Detection Tools

Over the past several years, a range of tools for cleaning data has appeared on the market and research groups have made available to the public software packages that can be used for duplicate record detection. In this section, we review such packages, focusing on tools that have open architecture and allow the users to understand the underlying mechanics of the matching mechanisms.

The Febrl system[3] (Freely Extensible Biomedical Record Linkage) is an open-source data cleaning toolkit, and it has two main components: The first component deals with data standardization and the second performs the actual duplicate detection. The data standardization relies mainly on hidden-Markov models (HMMs); therefore, Febrl typically requires training to correctly parse the database entries. For duplicate detection, Febrl implements a variety of string similarity metrics, such as Jaro, edit distance, and q-gram distance (see Section III). Finally, Febrl supports phonetic encoding (Soundex, NYSIIS, and Double Metaphone) to detect similar names. Since phonetic similarity is sensitive to errors in the first letter of a name, Febrl also

[3] http://sourceforge.net/projects/febrl

computes phonetic similarity using the reversed version of the name string, sidestepping the "first-letter" sensitivity problem.

TAILOR [30] is a flexible record matching toolbox, which allows the users to apply different duplicate detection methods on the data sets. The flexibility of using multiple models is useful when the users do not know which duplicate detection model will perform most effectively on their particular data. TAILOR follows a layered design, separating comparison functions from the duplicate detection logic. Furthermore, the execution strategies, which improve the efficiency are implemented in a separate layer, making the system more extensible than systems that rely on monolithic designs. Finally, TAILOR reports statistics, such as estimated accuracy and completeness, which can help the users understand better the quality of the a given duplicate detection execution over a new data set.

WHIRL[4] is a duplicate record detection system available for free for academic and research use. WHIRL uses the *tf.idf* token-based similarity metric to identify similar strings within two lists. The Flamingo Project[5] is a similar tools that provides a simple string matching tool that takes as input two string lists and returns the strings pairs that are within a prespecified edit distance threshold. WizSame by WizSoft is also a product that allows the discovery of duplicate records in a database. The matching algorithm is very similar to SoftTF.IDF (see Section III-B): two records match if they contain a significant fraction of identical or similar words, where similar are the words that are within edit distance one.

BigMatch [102] is the duplicate detection program used by the U.S. Census Bureau. It relies on blocking strategies to identify potential matches between the records of two relations, and scales well for very large data sets. The only requirement is that one of the two relations should fit in memory, and it is possible to fit in memory even relations with 100 million records. The main goal of BigMatch is not to perform sophisticated duplicate detection, but rather to generate a set of candidate pairs that should be then processed by more sophisticated duplicate detection algorithms.

Finally, we should note that currently many database vendors (Oracle, IBM, Microsoft) do not provide sufficient tools for duplicate record detection. Most of the efforts until now has focused

[4] http://www.cs.cmu.edu/~wcohen/whirl/

[5] http://www.ics.uci.edu/~flamingo/

on creating easy-to-use ETL tools, that can standardize database records and fix minor errors, mainly in the context of address data. Another typical function of the tools that are provided today is the ability to use reference tables and standardize the representation of entities that are well-known to have multiple representations. (For example, "TKDE" is also frequently written as "IEEE TKDE" or as "Transactions on Knowledge and Data Engineering.") A recent, positive step is the existence of multiple data cleaning operators within Microsoft SQL Server Integration Services, which is part of Microsoft SQL Server 2005. For example, SQL server now includes the ability to perform "fuzzy matches" and implements "error-tolerable indexes" that allow fast execution of such approximate lookups. The adopted similarity metric is similar to SoftTF.IDF, described in Section III-B. Ideally, the other major database vendors would also follow suit and add similar capabilities and extend the current ETL packages.

VII. Future Directions and Conclusions

In this survey, we have presented a comprehensive survey of the existing techniques used for detecting non-identical duplicate entries in database records. The interested reader may also want to read a complementary survey by Winkler [100] and the Special Issue of the IEEE Data Engineering Bulletin on Data Quality [45].

As database systems are becoming more and more commonplace, data cleaning is going to be the cornerstone for correcting errors in systems which are accumulating vast amounts of errors on a daily basis. Despite the breadth and depth of the presented techniques, we believe that there is still room for substantial improvements in the current state-of-the-art.

First of all, it is currently unclear which metrics and techniques are the current state-of-the-art. The lack of standardized, large scale benchmarking data sets can be a big obstacle for the further development of the field, as it is almost impossible to convincingly compare new techniques with existing ones. A repository of benchmark data sources with known and diverse characteristics should be made available to developers so they may evaluate their methods during the development process. Along with benchmark and evaluation data, various systems need some form of training data to produce the initial matching model. Although small data sets are available, we are not aware of large-scale, validated data sets that could be used as benchmarks. Winkler [98] highlights techniques on how to derive data sets that are properly anonymized and are still useful for duplicate record detection purposes.

Currently, there are two main approaches for duplicate record detection. Research in databases emphasizes relatively simple and fast duplicate detection techniques, that can be applied to databases with millions of records. Such techniques typically do not rely on the existence of training data, and emphasize efficiency over effectiveness. On the other hand, research in machine learning and statistics aims to develop more sophisticated matching techniques that rely on probabilistic models. An interesting direction for future research is to develop techniques that combine the best of both worlds.

Most of the duplicate detection systems available today offer various algorithmic approaches for speeding up the duplicate detection process. The changing nature of the duplicate detection process also requires adaptive methods that detect different patterns for duplicate detection and automatically *adapt themselves over time*. For example, a background process could monitor the current data, incoming data and any data sources that need to be merged or matched, and decide, based on the observed errors, whether a revision of the duplicate detection process is necessary or not. Another related aspect of this challenge is to develop methods that permit the user to derive the proportions of errors expected in data cleaning projects.

Finally, large amounts of structured information is now derived from unstructured text and from the web. This information is typically imprecise and noisy; duplicate record detection techniques are crucial for improving the quality of the extracted data. The increasing popularity of information extraction techniques is going to make this issue more prevalent in the future, highlighting the need to develop robust and scalable solutions. This only adds to the sentiment that more research is needed in the area of duplicate record detection and in the area of data cleaning and information quality in general.

REFERENCES

[1] Eugene Agichtein and Venkatesh Ganti. Mining reference tables for automatic text segmentation. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004)*, pages 20–29, 2004.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Searching with numbers. In *Proceedings of the 11th International World Wide Web Conference (WWW11)*, pages 420–431, 2002.

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1st edition, February 1993.

[4] Stephen F. Altschula, Warren Gisha, Webb Millerb, Eugene W. Meyersc, and David J. Lipmana. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

[5] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB 2002)*, 2002.

[6] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[7] Rohan Baxter, Peter Christen, and Tim Churches. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD '03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 25–27, 2003.

[8] Indrajit Bhattacharya and Lise Getoor. Latent dirichlet allocation model for entity resolution. Technical Report CS-TR-4740, Computer Science Department, University of Maryland, August 2005.

[9] Mikhail Bilenko, Raymond J. Mooney, William Weston Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, September/October 2003.

[10] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT' 98: Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100, 1998.

[11] Vinayak R. Borkar, Kaustubh Deshmukh, and Sunita Sarawagi. Automatic segmentation of text into structured records. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD 2001)*, pages 175–186, 2001.

[12] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. CRC Press, July 1984.

[13] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *Proceedings of the Sixth International World Wide Web Conference (WWW6)*, pages 1157–1166, 1997.

[14] Abhirup Chatterjee and Arie Segev. Data manipulation in heterogeneous databases. *ACM SIGMOD Record*, 20(4):64–68, December 1991.

[15] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD 2003)*, pages 313–324, 2003.

[16] Surajit Chaudhuri, Venkatesh Ganti, and Rajeev Motwani. Robust identification of fuzzy duplicates. In *Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE 2005)*, pages 865–876, 2005.

[17] Peter Cheeseman and John Sturz. Bayesian classification (autoclass): Theory and results. In *Advances in knowledge discovery and data mining*, pages 153–180. AAAI Press/The MIT Press, 1996.

[18] Junghoo Cho, Narayanan Shivakumar, and Hector Garcia-Molina. Finding replicated web collections. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD 2000)*, pages 355–366, 2000.

[19] Munir Cochinwala, Verghese Kurien, Gail Lalk, and Dennis Shasha. Efficient data reconciliation. *Information Sciences*, 137(1-4):1–15, September 2001.

[20] William W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems*, 18(3):288–321, 2000.

[21] William W. Cohen, Henry Kautz, and David McAllester. Hardening soft information sources. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, pages 255–259, 2000.

[22] William Weston Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*, pages 201–212, 1998.

[23] William Weston Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data

integration. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, 2002.

[24] David A. Cohn, Les Atlas, and Richard E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[25] Tamraparni Dasu, Theodore Johnson, Shanmugauelayut Muthukrishnan, and Vladislav Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 2002)*, pages 240–251, 2002.

[26] Arthur Pentland Dempster, Nan McKenzie Laird, and Donald Bruce Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B(39):1–38, 1977.

[27] Debabrata Dey, Sumit Sarkar, and Prabuddha De. Entity matching in heterogeneous databases: A distance based decision model. In *31st Annual Hawaii International Conference on System Sciences (HICSS'98)*, pages 305–313, 1998.

[28] Nelson S. D'Andrea Du Bois, Jr. A solution to the problem of linking multivariate documents. *Journal of the American Statistical Association*, 64(325):163–174, March 1969.

[29] Richard Oswald Duda and Peter Elliot Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[30] Mohamed G. Elfeky, Ahmed K. Elmagarmid, and Vassilios S. Verykios. TAILOR: A record linkage tool box. In *Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE 2002)*, pages 17–28, 2002.

[31] Ivan Peter Fellegi and Alan B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, December 1969.

[32] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. Declarative data cleaning: Language, model, and algorithms. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001)*, pages 371–380, 2001.

[33] Leicester E. Gill. OX-LINK: The Oxford medical record linkage system. In *Proceedings of the International Record Linkage Workshop and Exposition*, pages 15–33, 1997.

[34] Luis Gravano, Panagiotis G. Ipeirotis, Hosagrahar Visvesvaraya Jagadish, Nick Koudas, Shanmugauelayut Muthukrishnan, Lauri Pietarinen, and Divesh Srivastava. Using q-grams in a DBMS for approximate string processing. *IEEE Data Engineering Bulletin*, 24(4):28–34, December 2001.

[35] Luis Gravano, Panagiotis G. Ipeirotis, Hosagrahar Visvesvaraya Jagadish, Nick Koudas, Shanmugauelayut Muthukrishnan, and Divesh Srivastava. Approximate string joins in a database (almost) for free. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001)*, pages 491–500, 2001.

[36] Luis Gravano, Panagiotis G. Ipeirotis, Nick Koudas, and Divesh Srivastava. Text joins in an RDBMS for web data integration. In *Proceedings of the 12th International World Wide Web Conference (WWW12)*, pages 90–101, 2003.

[37] Sudipto Guha, Nick Koudas, Amit Marathe, and Divesh Srivastava. Merging the results of approximate match operations. In *Proceedings of the 30th International Conference on Very Large Databases (VLDB 2004)*, pages 636–647, 2004.

[38] Trevor Hastie, Robert Tibshirani, and Jerome Harold Friedman. *The Elements of Statistical Learning*. Springer Verlag, August 2001.

[39] Mauricio Antonio Hernández and Salvatore Joseph Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, January 1998.

[40] Matthew A. Jaro. Unimatch: A record linkage system: User's manual. Technical report, U.S. Bureau of the Census, Washington, D.C., 1976.

[41] Matthew A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):414–420, June 1989.

[42] Thorsten Joachims. Making large-scale SVM learning practical. In Bernhard Schölkopf, Christopher John C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT-Press, 1999.

[43] Ralph Kimball and Joe Caserta. *The data warehouse ETL toolkit: Practical techniques for extracting, cleaning, conforming, and delivering data*. John Wiley & Sons, 2004.

[44] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *Proceedings of the 14th International Conference on Machine Learning (ICML'97)*, pages 170–178, 1997.

[45] Nick Koudas, editor. *IEEE Data Engineering Bulletin*, volume 29. IEEE, June 2006. Special Issue on Data Quality.

[46] Nick Koudas, Amit Marathe, and Divesh Srivastava. Flexible string matching against large databases in practice. In *Proceedings of the 30th International Conference on Very Large Databases (VLDB 2004)*, pages 1078–1086, 2004.

[47] Karen Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, December 1992.

[48] Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, June 1989.

[49] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965. Original in Russian – translation in Soviet Physics Doklady 10(8):707-710, 1966.

[50] Ee-Peng Lim, Jaideep Srivastava, Satya Prabhakar, and James Richardson. Entity identification in database integration. In *Proceedings of the Ninth IEEE International Conference on Data Engineering (ICDE 1993)*, pages 294–301, 1993.

[51] Nikos Mamoulis. Efficient processing of joins on set-valued attributes. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD 2003)*, pages 157–168, 2003.

[52] Andrew McCallum. Information extraction: Distilling structured data from unstructured text. *ACM Queue*, 3(9):48–57, 2005.

[53] Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 591–598, 2000.

[54] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, pages 169–178, 2000.

[55] Andrew McCallum and Ben Wellner. Conditional models of identity uncertainty with application to noun coreference. In *Advances in Neural Information Processing Systems (NIPS 2004)*, 2004.

[56] Ruslan Mitkov. *Anaphora Resolution*. Longman, 1st edition, August 2002.

[57] Alvaro E. Monge and Charles P. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 267–270, 1996.

[58] Alvaro E. Monge and Charles P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the 2nd ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'97)*, pages 23–29, 1997.

[59] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[60] Saul Ben Needleman and Christian Dennis Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[61] Howard B. Newcombe. Record linking: The design of efficient systems for linking records into individual and family histories. *American Journal of Human Genetics*, 19(3):335–359, May 1967.

[62] Howard B. Newcombe. *Handbook of Record Linkage*. Oxford University Press, 1988.

[63] Howard B. Newcombe and James M. Kennedy. Record linkage: Making maximum use of the discriminating power of identifying information. *Communications of the ACM*, 5(11):563–566, November 1962.

[64] Howard B. Newcombe, James M. Kennedy, S.J. Axford, and A.P. James. Automatic linkage of vital records. *Science*, 130(3381):954–959, October 1959.

[65] Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom M. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

[66] Hanna Pasula, Bhaskara Marthi, Brian Milch, Stuart J. Russell, and Ilya Shpitser. Identity uncertainty and citation matching. In *Advances in Neural Information Processing Systems (NIPS 2002)*, pages 1401–1408, 2002.

[67] Mike Perkowitz, Robert B. Doorenbos, Oren Etzioni, and Daniel Sabey Weld. Learning to understand information on the Internet: An example-based approach. *Journal of Intelligent Information Systems*, 8(2):133–153, March 1997.

[68] Lawrence Philips. Hanging on the metaphone. *Computer Language Magazine*, 7(12):39–44, December 1990. Accessible at `http://www.cuj.com/documents/s=8038/cuj0006philips/`.

[69] Lawrence Philips. The double metaphone search algorithm. *C/C++ Users Journal*, 18(5), June 2000.

[70] Jose C. Pinheiro and Don X. Sun. Methods for linking and mining heterogeneous databases. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 309–313, 1998.

[71] Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001)*, pages 381–390, 2001.

[72] Pradeep Ravikumar and William Weston Cohen. A hierarchical graphical model for record linkage. In *20th Conference on Uncertainty in Artificial Intelligence (UAI 2004)*, 2004.

[73] Eric Sven Ristad and Peter N. Yianilos. Learning string edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, May 1998.

[74] Elke Rundensteiner, editor. *IEEE Data Engineering Bulletin*, volume 22. IEEE, January 1999. Special Issue on Data Transformation.

[75] Robert C. Russell. Index, U.S. patent 1,261,167. Available at `http://patft.uspto.gov/netahtml/srchnum.htm`, April 1918.

[76] Robert C. Russell. Index, U.S. patent 1,435,663. Available at `http://patft.uspto.gov/netahtml/srchnum.htm`, November 1922.

[77] Sunita Sarawagi, editor. *IEEE Data Engineering Bulletin*, volume 23. IEEE, December 2000. Special Issue on Data Cleaning.

[78] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, pages 269–278, 2002.

[79] Sunita Sarawagi and Alok Kirpal. Efficient set joins on similarity predicates. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD 2004)*, pages 743–754, 2004.

[80] Parag Singla and Pedro Domingos. Multi-relational record linkage. In *KDD-2004 Workshop on Multi-Relational Data Mining*, pages 31–48, 2004.

[81] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[82] Aya Soffer, David Carmel, Doron Cohen, Ronald Fagin, Eitan Farchi, Michael Herscovici, and Yoelle S. Maarek. Static index pruning for information retrieval systems. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2001*, pages 43–50, 2001.

[83] Erkki Sutinen and Jorma Tarhio. On using q-gram locations in approximate string matching. In *Proceedings of Third Annual European Symposium on Algorithms (ESA'95)*, pages 327–340, 1995.

[84] Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of the 21st International Conference on Machine Learning (ICML 2004)*, 2004.

[85] Robert L. Taft. Name search techniques. Technical Report Special Report No. 1, New York State Identification and Intelligence System, Albany, NY, February 1970.

[86] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning object identification rules for information integration. *Information Systems*, 26(8):607–633, 2001.

[87] Sheila Tejada, Craig Alan Knoblock, and Steven Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, 2002.

[88] Benjamin J. Tepping. A model for optimum linkage of records. *Journal of the American Statistical Association*, 63(324):1321–1332, December 1968.

[89] Esko Ukkonen. Approximate string matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, 1992.

[90] Julian R. Ullmann. A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words. *The Computer Journal*, 20(2):141–147, 1977.

[91] Vassilios S. Verykios, Ahmed K. Elmagarmid, and Elias N. Houstis. Automating the approximate record matching process. *Information Sciences*, 126(1-4):83–98, July 2000.

[92] Vassilios S. Verykios and George V. Moustakides. A generalized cost optimal decision model for record matching. In *Proceedings of the 2004 International Workshop on Information Quality in Information Systems*, pages 20–26, 2004.

[93] Vassilios S. Verykios, George V. Moustakides, and Mohamed G. Elfeky. A bayesian decision model for cost optimal record matching. *VLDB Journal*, 12(1):28–40, May 2003.

[94] Y. Richard Wang and Stuart E. Madnick. The inter-database instance identification problem in integrating autonomous systems. In *Proceedings of the Fifth IEEE International Conference on Data Engineering (ICDE 1989)*, pages 46–55, 1989.

[95] Michael S. Waterman, Temple F. Smith, and William A. Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20(4):367–387, 1976.

[96] Jennifer Widom. Research problems in data warehousing. In *Proceedings of the 1995 ACM Conference on Information and Knowledge Management (CIKM'95)*, pages 25–30, 1995.

[97] William E. Winkler. Improved decision rules in the Felligi-Sunter model of record linkage. Technical Report Statistical Research Report Series RR93/12, U.S. Bureau of the Census, Washington, D.C., 1993.

[98] William E. Winkler. The state of record linkage and current research problems. Technical Report Statistical Research Report Series RR99/04, U.S. Bureau of the Census, Washington, D.C., 1999.

[99] William E. Winkler. Methods for record linkage and bayesian networks. Technical Report Statistical Research Report Series RRS2002/05, U.S. Bureau of the Census, Washington, D.C., 2002.

[100] William E. Winkler. Overview of record linkage and current research directions. Technical Report Statistical Research Report Series RRS2006/02, U.S. Bureau of the Census, Washington, D.C., 2006.

[101] William E. Winkler and Yves Thibaudeau. An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census. Technical Report Statistical Research Report Series RR91/09, U.S. Bureau of the Census, Washington, D.C., 1991.

[102] William E. Yancey. Bigmatch: A program for extracting probable matches from a large file for record linkage. Technical Report Statistical Research Report Series RRC2002/01, U.S. Bureau of the Census, Washington, D.C., March 2002.

[103] William E. Yancey. Evaluating string comparator performance for record linkage. Technical Report Statistical Research Report Series RRS2005/05, U.S. Bureau of the Census, Washington, D.C., June 2005.

[104] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490, December 1998.