Schema Matching using Duplicates

Alexander Bilke
Technische Universität Berlin, Germany
bilke@cs.tu-berlin.de

Felix Naumann
Humboldt-Universität zu Berlin, Germany
naumann@informatik.hu-berlin.de

Abstract

Most data integration applications require a matching between the schemas of the respective data sets. We show how the existence of duplicates within these data sets can be exploited to automatically identify matching attributes. We describe an algorithm that first discovers duplicates among data sets with unaligned schemas and then uses these duplicates to perform schema matching between schemas with opaque column names.

Discovering duplicates among data sets with unaligned schemas is more difficult than in the usual setting, because it is not clear which fields in one object should be compared with which fields in the other. We have developed a new algorithm that efficiently finds the most likely duplicates in such a setting. Now, our schema matching algorithm is able to identify corresponding attributes by comparing data values within those duplicate records. An experimental study on real-world data shows the effectiveness of this approach.

1 Introduction

Integrating data from heterogeneous sources yields many difficult problems of technical, structural, and semantic nature. Technical challenges, such as remote access to sources using different transfer protocols, are usually considered to be solved, for instance using wrapper technologies. Structural challenges include translation between different query languages and data models, integrating schemas, and processing queries, to name a few. Among the most daunting challenges, however, are those concerning semantics, because implicit knowledge known only by human experts must be made explicit. Herein, we tackle the well-known problem of *schema matching*, i.e., the task of identifying semantically same or similar elements in two different schemas. Schema matching techniques are used to determine correspondences between heterogeneous schemas. Correspondences, in turn, are used to discover mappings and queries that transform data from one database to data conforming to the schema of the other database [26]. We call schemas with a mapping between them *aligned*.

In general, schema matching can be performed by comparing the names of the attributes, by comparing properties of underlying data instances, by comparing the structure in which an attribute is embedded, or any combination of the three [28]. In the DUMAS (Duplicate-based Matching of Schemas) project[1] we approach the schema matching problem by designing an instance-based matching algorithm. Usual instance-based approaches analyze attributes of each schema individually, extracting properties about the attributes, such as distribution of characters, average string length, etc. Attributes having similar properties are subsequently matched, i.e., they are assumed to have the same meaning. We call such approaches *vertical matching*, because properties of columns of tables are compared. The DUMAS approach, on the other hand, performs *horizontal matching*: We traverse tables in search for similar rows (or tuples), in effect detecting duplicates. Once a few duplicates have been discovered, deriving a schema matching is simple in principle: Same or similar data values among the duplicates imply corresponding attributes of the schemas. This horizontal approach must solve two main problems: (i) Detecting duplicates among databases with opaque schemas and (ii) deriving a schema matching from a set of fuzzy duplicates.

Duplicate detection is the problem of identifying multiple representations of the same real-world object within a set of objects. Such multiple representations are called *fuzzy* or *approximate duplicates*, because they might not be exact copies of one another. In the remainder we simply use the term *duplicates*. Duplicates arise during data creation, where they are inadvertently generated, and during data integration if multiple sources store data about same real world objects.

Finding duplicates among two databases with unaligned schemas is more difficult than the classical duplicate detection problem (in the field of data cleansing also known as object identification or record linkage [29]). There, du-

[1] " 'And you, count, have made this match?' asked Beauchamp."
(Alexandre Dumas "The Count of Monte Cristo")

plicates are searched within a single table, so it is already clear which data values among a pair of tuples to compare. Typical approaches use domain-specific rules based on such comparisons to determine whether a pair of tuples represents the same real-world object. In our case, the two tuples are from different and heterogeneous tables, so they possibly have only a subset of their attributes in common. Also, those attributes that they do have in common do not necessarily appear in the same order. Finally, same data values might appear in different contexts in different tuples, thus misleading duplicate detection. For instance, "Umeshwar Dayal" is an editor of the ICDE 2003 proceedings but also author of numerous articles in ICDE and elsewhere. To compensate, we present a duplicate detection algorithm that does not rely on schema information and detects duplicates based only on the sets of data values that the tuples comprise.

Research on the classical duplicate detection problem is further concerned with *efficiently* finding all duplicates by reducing the number of tuple comparisons while still maintaining good detection levels. Reduction is usually achieved by partitioning tuples based on application-specific criteria and searching for duplicates only within the partitions. In our case, such a partitioning is not possible, because we do not know the semantics of the attributes. Instead, our duplicate detection algorithm uses a similarity measure for which techniques are known that efficiently produce the top K results.

Once such duplicates have been found, we use them to drive *schema matching*. Although the basic idea is simple—same attribute values in a duplicate imply same attributes in the schemas—we must overcome several subtle problems: Due to misspellings, different formats, and other discrepancies (i.e., due to fuzziness), duplicates often do not have same but merely similar attribute values. Thus, we must decide how similar attribute values must be and how many duplicates with such similar values are needed to confidently derive an attribute match. Also, in real-world examples different attributes within a tuple may have the same value. For instance, shipping address and billing address often have the same value in a record but have different meaning. Thus, even given some duplicates, finding corresponding attributes is not always trivial. In this paper we present a procedure to perform schema matching using a set of fuzzy duplicates. The matching can be iteratively refined by on-demand repetition of duplicate detection.

Contributions. Our main contribution is a new and effective *instance-based method for schema matching*. The key feature of this method is that it does not rely on any kind of attribute names in either of the schemas. Also, it is an improvement over other instance-based schema matching techniques, because it recognizes syntactically similar but semantically different attributes. An additional contribution is a new *method for duplicate detection* that does not make the usual assumption of duplicates having the same schema, or even of having any known schema.

Assumptions. Like all instance-based schema matching approaches, we assume that both schemas have some underlying data. Schema matching is usually performed either to assist integration of sources into a newly integrated schema (e.g., in a data warehouse) or to assist mapping between two data sources (e.g., in a peer data management system). While in the latter case both sources have underlying data, the former case fulfills the assumption only after at least one source has been used to populate the integrated schema. Nevertheless, even there schema matching greatly reduces the burden of assigning correspondences for many schemas by hand.

What is more (and unlike other approaches), we assume the existence of at least some duplicates among the sources. While this assumption is certainly not always fulfilled, there are many scenarios where this is the case: Some customer having ordered at two different online stores; same books sold at several stores; an apartment offered at several web sites; the expression of a gene stored in many genomic databases; etc. Experiments have shown that only very few duplicates are needed to successfully perform schema matching, reducing the weight of this assumption.

Structure of this paper. Related work in the areas of duplicate detection and schema matching is discussed in Section 2. Section 3 elaborates the main idea with a simple example of two small databases, a few duplicates, and some corresponding attributes. Section 4 introduces the new similarity measure to detect duplicates in databases with opaque schemas. These duplicates are input to the schema matching method presented in Section 5. Section 6 describes experimental results on artificial and on real-world data before we conclude in Section 7.

2 Related work

Because our work combines techniques of two major research topics in a single approach, related work draws from two areas: duplicate detection and schema matching.

2.1 Duplicate detection

The problem of identifying different representations of the same real world object comes in many guises: Duplicate detection in data cleansing frameworks [17, 1], record linkage [14, 31, 13], entity or object identification [18], etc. Rahm and Do provide a comprehensive overview [29].

Almost all approaches assume aligned and possibly integrated schemas, and thus conceptually reduce the problem to finding duplicates in a single table. In the *merge/purge*

method of Hernández and Stolfo duplicates within a single table are detected using application-specific rules [17]. To reduce computation complexity, the table is sorted using a domain-specific key, and only records within a sliding window are compared. In order to improve precision, Ananthakrishna et al. also exploit the hierarchical structure to detect duplicates within a single dimension of a data warehouse [1]. *Record linkage* follows a probabilistic approach [31, 13]. For each record pair, a comparison vector is produced by comparing corresponding attribute values. The record pairs are classified as *matched*, *possibly matched*, and *unmatched* using a linkage rule that assigns each observed comparison vector with a probability for each class. To reduce the number of comparisons, application-specific blocking criteria can be used. Only Doan et al. consider the problem of duplicate detection with only partially aligned schemas [12]. In fact, the authors make explicit use of non-aligned attribute values to improve duplicate detection. Nevertheless, the approach assumes that the existing alignment is known in advance.

Several recent approaches incorporate machine learning into the duplicate detection process. Tejada et al. use a decision tree forest to learn both duplicate detection rules and weights for string transformations, which are used for comparing fields [30]. The string edit distance is a metric commonly used in duplicate detection procedures. Bilenko and Mooney have shown that machine learning techniques increase the accuracy of the field matching task when string edit distance is used, and in some cases even when token-based measures are used [4].

All methods described above require the schemas under consideration to be aligned. This is necessary to reduce the number of field-wise comparisons and to define meaningful comparison rules. In this paper we drop this assumption because our goal is to find the attribute correspondences. Instead, we consider the problem of duplicate detection with unknown attribute semantics and thus with no alignment among schemas.

2.2 Schema matching

As described above, known duplicate detection methods require the schemas under consideration to be aligned. *Schema matching* is the task of finding such an alignment, i.e., finding attribute correspondences between two given schemas. In a greater context, schema matching can be seen as a complex operator *Match* in a model management application [2]. Several semi-automatic solutions to the schema matching problem have been proposed (see survey of Rahm and Bernstein [28]). Schema-based methods primarily use labels, types, and other schema information to find similar attributes [20, 21]. Instance-based methods also exploit properties of the underlying data, possibly in combination with attribute labels, to derive matches [11, 23]. As opposed to most solutions of the latter category, which use summary information for attribute classification, we derive a schema matching from identified duplicates in the databases.

We know of three approaches that perform schema matching after duplicate detection. The *Internet Learning Agent* (ILA) [25] establishes correspondences between objects of its known internal world model and an external information source by choosing an object in the internal model and trying to find a duplicate in the information source. Duplicates are identified as a pair of objects that have at least one common data value. Due to this untargeted approach, the ILA relies on high extensional overlap: The approach assumes "spanning sources" covering the entire information domain. While such sources may exist in a few select application domains, such as the mentioned whois service, the assumption generally is unrealistic. In our approach, we assume only a very small extensional overlap. Furthermore, ILA considers two attributes related when their values match *exactly*, thus, ignoring different representations of the same information.

Chua et al. perform statistical analysis of the data in duplicates to derive matches between the respective records [5]. The focus of their work is to detect correlations among differently scaled and encoded numerical attributes using duplicates, whereas the work described in this paper is mostly concerned with textual data. Duplicates are assumed to be identified through a common ID attribute; thus, the authors assume at least this one attribute to already be aligned. Furthermore, entity identifiers that are consistently used in all databases exist in only few cases.

The schema matching system iMAP [8] detects correspondences using both schema and instance information. Its use of data falls into the category of vertical matchers. But in the case where duplicates exist, special "overlap" modules replace the standard modules. However, the duplicates used by these modules are not automatically detected and must be provided by the user. Furthermore, only exact matches on attribute values are considered by the overlap text searcher when looking for attribute correspondences.

Our solution to schema matching drops several of the assumptions made in other work, but also has certain restrictions. In particular, it is only applicable in the presence of an extensional overlap, i.e., at least a small number of real world objects must be represented in both databases. Our solution is complementary to the solutions described above, and we regard it as a fine candidate for inclusion in a composite schema matching system such as COMA [10].

3 Motivating example

For simplicity, we discuss the problem and formulate our solutions using the relational data model: Let R and S be

R	A	B	C	D	E
r_1	John	Doe	m	(408) 7573339	(408) 7573338
r_2	Joe	Smith	m	(249) 3615616	(249) 2342366
r_3	Suzy	Klein	f	(358) 2436321	(358) 2436321
r_4	Sam	Adams	m	(541) 8127100	(541) 8121164
r_5	Mark	Spitz	m	(901) 8319311	(901) 8612382
r_6	Jim	Beam	-	(782) 1238957	(781) 1883744
r_7	Kate	Moss	f	(124) 9654565	-
r_8	Sam	Wong	f	(124) 4955670	(999) 9999999
r_9	John	Dean	m	(369) 3663624	(367) 3663625

S	B'	F	E'	G
s_1	Doe	jdoe	408-9182043	XP
s_2	Deen	jdean	369-3663625	XP
s_3	Klein	suzy	358-2436321	UNIX
s_4	Adams	adams	541-8121164	W2000
s_5	Wong	kate	923-6363443	Linux
s_6	Kurz	itsme	-	UNIX

Figure 1. Relations R and S with intensional and extensional overlap.

two relations R(A, B, C, D, E) and S(B', F, E', G). There is some, yet unknown intensional overlap (corresponding attributes). We deliberately chose meaningless attribute names to reflect real-world situations where attribute names are missing, do not carry a meaning, or carry an unknown meaning. In the example, corresponding attributes have same letters as names; real-world names of corresponding attributes can be widely different. Further, let R and S have tuples $r_1, \ldots, r_9$ and $s_1, \ldots, s_6$. R and S have some yet unknown extensional overlap, i.e., some duplicates. This simple scenario is shown in Figure 1 (next page).

The main goal addressed in this paper is to automatically find a correct schema matching as shown in Figure 2. Note that the intensional overlap of R and S is only partial, so not all attributes have a matching partner. Also note again that similar attribute names are chosen only for presentational reasons; we solve the problem of schema matching with meaningless or opaque names.

Attr. of R		Attr. of S
A	$\longleftrightarrow$	–
B	$\longleftrightarrow$	B'
C	$\longleftrightarrow$	–
D	$\longleftrightarrow$	–
E	$\longleftrightarrow$	E'
–	$\longleftrightarrow$	F
–	$\longleftrightarrow$	G

Figure 2. The correct schema matching from R to S

Close examination of R and S in Figure 1 reveals an extensional overlap: Tuple-pairs (r_3, s_3), (r_4, s_4), and (r_9, s_2) are duplicates, which we can use to perform schema matching. Figure 3 shows tuples r_3 and s_3 with indications of attribute matches based on equal or similar data values.

From this single duplicate, we can already derive useful clues for matching. In Section 5 we formally encode these "clues" in a similarity matrix:

- B $\longleftrightarrow$ B' is a perfect match with respect to this duplicate, because B and B' have the same value.

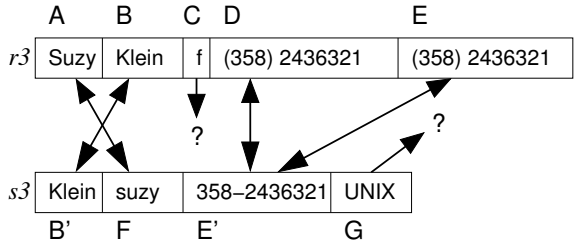

Figure 3. Using a duplicate to derive some attribute matches

- A $\longleftrightarrow$ F is also a perfect match, because the values in A and F are equal (we ignore upper/lower cases).
- Both C and G are not likely to participate in any match, because their values have no same or similar equivalent in the other tuple. Remember that not all attributes must participate in a match.
- Both D $\longleftrightarrow$ E' and E $\longleftrightarrow$ E' are possible matches, because the value in E' is similar to the values both in D and E. In the final result, we allow only 1:1 matches, so we will have to decide which of the two matches is correct.

By finding more duplicates and repeating the discovery of same and similar values, we are able to refine the overall matching. For instance, by examining the duplicate (r_4, s_4) we

- confirm the match B $\longleftrightarrow$ B', because 'Adams' = 'Adams',
- foster a doubt for match A $\longleftrightarrow$ F, because the values 'Sam' and 'adams' differ,
- confirm that C and G have no match, because again they have no similar values, and
- shift our decision towards E $\longleftrightarrow$ E' and away from D $\longleftrightarrow$ E', because '(541) 8121164' and '541-8121164' are quite similar, but '(541) 8127100' and

'541-8121164' are much less so, using some suitable similarity measure.

Of course, real-life scenarios are not always as straightforward. First, duplicates are usually only fuzzy, so we must rely on a similarity function to deduce duplicates. Second, low intensional overlap can mislead duplicate detection into believing that no duplicates exist. Third, low extensional overlap may result in an insufficient number of duplicates to deduce attribute matches with enough certainty. Fourth, same or similar values do not always imply matching attributes; for instance, an author in one publication record could be an editor in another, or first and last names are chosen as user accounts as in the example of Figure 3. In the following sections we address these difficulties and describe how to overcome them.

4 Duplicate detection with unaligned schemas

As we show in Section 5, knowledge about duplicates can be exploited for schema matching. However, finding these duplicates is difficult precisely because the schemas are not yet aligned. In this section we present an algorithm that uses a domain-independent measure for comparing two tuples of unknown and possibly heterogeneous structure. Recall that our goal is to find as many expressive duplicates as necessary for schema matching, in contrast to the usual problem of finding *all* duplicates in a data set. Our duplicate detection algorithm achieves this goal by detecting the K most similar tuple pairs. The comparison measure must overcome four problems:

1. *Unknown schema alignment*: It is unclear which field in one tuple to compare with which field in the other.

2. *Unknown attribute semantics*: We cannot make use of domain knowledge to formulate an effective comparison measure. Common duplicate detection methods use manually or statistically created rules that are based on the similarity of certain corresponding attributes. Without such matches, meaningful rules cannot be created. Instead, a comparison measure that is independent of the fields' semantics must be applied.

3. *Misleading value similarities*: Attribute values of non-corresponding attributes could coincidentally be similar, although their respective tuples are not duplicates. Consider tuples r_7 and s_5 in Figure 1: Both tuples have an attribute value 'kate', but are not duplicates. Without knowledge of the correct attribute matches, such a value match can mislead duplicate detection.

4. *Partial schema overlap*: Not all fields in one tuple necessarily have a matching partner in the other. With only few corresponding attributes, the similarity of two tuples is typically low. In our example, only two attributes among the relations actually match (B $\longleftrightarrow$ B' and E $\longleftrightarrow$ E' in Figure 2).

Our algorithm considers each tuple as a single string and employs a string comparison metric to compare two tuples (strings). After comparing two tuples, one has to determine if the two tuples are duplicates. Because other approaches assume same schema of the tuples, they can predefine a reasonable similarity threshold to identify duplicate tuples. Lacking this assumption, a fixed threshold is unreasonable: Similarity values vary with the degree of schema overlap. Instead, the algorithm ranks tuple pairs by their similarity and identifies the K most similar tuple pairs. We consider these to be duplicates—no threshold is needed.

Although not explicitly used in our experiments, data cleaning can improve the accuracy of duplicate detection and improve the performance. Some operations can be performed without knowledge of the attributes' semantics, e.g., the removal of stopwords and stemming [27]. Other procedures depend on the meaning of certain attributes, but can still be undertaken without knowing any attribute correspondences. In particular, representational issues can be resolved, e.g., by using standard encodings for date, gender, country, etc.

4.1 String comparison methods

According to the classification of Cohen et al., string comparison metrics fall into three categories: edit-distance like functions, token-based similarity measures, and hybrid similarity measures [7].

Edit-distance models determine the distance between two strings based on the minimal number of edit operations (insert, delete, replace) needed to transform one string into the other. Variants that allow specific costs for different character substitutions and for starting or continuing a gap have been reported [7]. Using a simple edit-distance function in our situation is impractical, because it assumes all attributes to have a matching partner (ignoring Problem 4) and it assumes attributes to appear in the same order (ignoring Problem 1). Deviations from these assumptions would unduly increase the distance and make duplicates indistinguishable. A block-edit-distance model allows for arbitrary movements of entire substrings, thus accounting for Problem 1. However, such models are in general very expensive [19]. We do not use edit-distance for duplicate detection, but we use it later to perform schema matching.

Token-based similarity measures interpret a string as a bag of words. The *vector space model* is widely used in the information retrieval community. In this model, each string is represented as a vector containing weights for each term of a (global) vocabulary. The similarity of two strings

is then determined with the *cosine measure*, which is the product of two vectors normalized to unit length, and thus, equal to the cosine of the angle between the two vectors. An advantage of the vector space model over an edit-distance model is its independence of term ordering. Therefore, it is used as the model for comparing tuples in the duplicate detection step.

Hybrid similarity measures are functions that use another metric as a secondary measure. An example for this group is the SoftTFIDF measure, which we use in the schema matching step for comparing attribute values (see Section 5 for details). Another hybrid measure is the recursive field matching algorithm [22]. Using this algorithm, each source attribute would be compared to each target attribute using the secondary similarity measure (e.g., edit distance). We have decided against this hybrid measure for various reasons, including efficiency of computation.

4.2 Similarity of unaligned tuples

In our implementation we use the cosine measure, i.e., we tokenize the tuples and compare the resulting vector representations. The assignment of weights for the tokens in each tuple is crucial for the effectiveness of the cosine measure: A weight should represent the relative importance of a token within the tuple. The well-known *TFIDF weighting scheme* calculates the weight as a function of the *term frequency (TF)*, i.e., the number of times the term occurs in the string, and the *inverse document frequency (IDF)*, which is the overall number of strings (tuples) divided by the number of strings in which the given term occurs. We define the weight $w'(s,t)$ of a term t in a string s as

$$w'(s,t) = \log(tf_{s,t} + 1) \cdot \log(\frac{N}{df_t} + 1) \qquad (1)$$

where $tf_{s,t}$ is the term frequency of t in s, N is the overall number of tuples, and df_t is the number of tuples in which t appears. These weights are normalized such that their respective vector has unit length. The *tuple similarity* of two tuples r and s is then calculated as

$$tupsim(r,s) = \sum_{t \in r \cap s} w(r,t) \cdot w(s,t) \qquad (2)$$

where w is the normalized weight. Using a TFIDF-based measure has several advantages. First, it is order-independent, which is important with respect to Problem 1. Second, by using the inverse document frequency, terms that occur in only few tuples receive a higher weight. Intuitively, the reason is that infrequent terms have a higher identifying power, and thus, should have a higher influence on the similarity score. This behavior can be of help in Problem 3.

4.3 Efficiently finding similar tuples

One major issue that has to be addressed is the complexity of this approach, i.e., the number of tuple pairs that must be compared. Comparing each record from the first database with each record in the second database is clearly infeasible. A naïve improvement is to compare only tuples that have at least one term in common: In Equation (2) only terms that have non-zero weight in *both* databases affect the similarity score. An inverted index on one of the databases can be used for that purpose. This approach does not work well with attributes that draw values from a very small domain (e.g., gender, grade, etc.), which would result in many tuples having at least one term in common, if the two schemas share such an attribute. However, terms that occur in many tuples have a low inverse document frequency, and thus, little effect on the similarity score.

To overcome this problem of comparing tuples because of common tokens with low weight, we use the Whirl algorithm [6]. It performs a focused search based on those common values that have high TFIDF weight and thus are more likely to be contained in the final result set: Within an A* search scheme Whirl calculates upper bounds for each tuple of one database and refines these bounds by combining those tuples with promising tuples of the other database. Tuples are promising if they have a common term with a high TFIDF weight.

The advantage of Whirl, apart from being very efficient, is that it is incremental: The current search state can be saved, and search can be resumed at a later point in time. We exploit this feature later, if the first set of duplicates is not sufficient to derive a schema matching with some certainty (see section 5.2).

Note that the algorithm consumes only a fraction of the database size in main memory. In cases where the size of the databases is too large, a sampling-based method such as the one described by Gravano et al. [16] can be used.

5 Schema matching with duplicates

Schema matching is the problem of identifying schema elements (here: attributes) that have same or similar semantics in two different schemas. Not all attributes necessarily have a corresponding partner. Also, there might not be only simple (1:1) matches, but complex matches relating one or several attributes in one schema with multiple attributes in the other schema. In this paper, we only consider the former case in the schema matching step. The goal of this step is to deduce attribute matches from K high-confidence duplicate pairs, which have been discovered in the duplicate detection step.

Each duplicate pair potentially provides some information for the construction of attribute matches: If two field

values are equal or highly similar, it is likely that their respective attributes correspond. In a first step we perform field-wise similarity comparisons for each of the K duplicates to generate a similarity matrix. Then we describe how use that matrix to arrive at an overall schema matching.

5.1 Similarity matrix

Given a duplicate pair $r = (a_1, \ldots, a_m)$ and $s = (b_1, \ldots, b_n)$, this step produces a $m \times n$ matrix that stores the similarity *fieldsim* of each pair of field values a_i and b_j in the tuples. For comparing tuple fields we use the *Soft-TFIDF measure* [7], a variation of the previously described TFIDF measure that also considers similar terms (as opposed to equal terms). We define $\text{CLOSE}(\theta, a_i, b_j)$ as the set of all terms in one field a_i that have a similar enough ($> \theta$) term in the other field:

$$\text{CLOSE}(\theta, a_i, b_j) := \{t_a \in a_i | \exists t_b \in b_j, termsim(t_a, t_b) > \theta\}. \quad (3)$$

Here, $termsim(t_a, t_b)$ is a secondary similarity function based on the normalized edit distance (*ed*):

$$termsim(t_a, t_b) := 1 - \frac{ed(t_a, t_b)}{\max\{|t_a|, |t_b|\}}. \quad (4)$$

where $\max\{|t_a|, |t_b|\}$ is the length of the longer term. Put together, the we can define SofTFIDF, called *fieldsim* in this paper:

$$\textit{fieldsim}(a_i, b_j) := \sum_{t_a \in \text{CLOSE}(\theta, a_i, b_j)} w(a_i, t_a) \cdot w(b_j, t_b) \cdot termsim(t_a, t_b) \quad (5)$$

where t_b is the term in b_j that is most similar to t_a.

Again, using SoftTFIDF has the advantage of order-independence. If one field contains the value "Joe Smith", and the other contains the value "Smith, Joe", they will be considered very similar. Computing SoftTFIDF is more expensive than computing TFIDF or edit distance. But because we expect to use only very few duplicates, the added complexity is negligible.

Consider tuples r_3 and s_3 of Figure 1. In the previous section we have discovered them to be duplicates. Table 1 shows their similarity matrix with values $\textit{fieldsim}(a_i, b_j)$ in each field.

We generate such a matrix M_k for each of the K duplicates and combine all matrices to produce the overall average similarity matrix M as input to the next step:

$$M = \frac{1}{K} \sum_{k=1}^{K} M_k \quad (6)$$

In summary, M stores average similarity scores that are accumulated over the field-similarities of the K most similar duplicates.

	Suzy	Klein	f	(358)2436321	(358)2436321
Klein	0	1.0	0	0	0
suzy	1.0	0	0	0	0
358-2436321	0	0	0	0.67	0.67
UNIX	0	0.2	0	0	0

Table 1. Field-similarity matrix for a duplicate pair

5.2 Matching attributes

Given the matrix M with similarity scores, the goal of this step is to derive an overall schema matching. First, we apply a user-defined threshold to M, setting all similarity values below the threshold to zero. Applying the threshold before finding the matching accommodates the attributes in either schema that do not have a matching partner. We then use the matrix as input to the bipartite weighted matching problem, also known as the *assignment problem* [24]. The optimal solution to this problem is a matching with the maximal sum of similarities and can be computed in polynomial time [15].

We classify the individual matches of this maximum matching into three classes: certain matches, possible matches, and non-matches. Non-matches are those that have a similarity below a user-specified threshold. To determine whether a match is *certain*, we calculate its contribution to the overall solution: We set its similarity score to zero, solve the assignment problem again, and compare this solution with the original one. If the difference is large, i.e., there is no alternate matching with a similar score, we call it certain. All other matches are classified as *possible*.

Table 2 shows matrix M for the duplicate pairs (r_3, s_3), (r_4, s_4), and (r_9, s_2). The maximal matching is shown in bold. Of this matching D $\longleftrightarrow$ G is a non-match and will be automatically removed because its value is below a threshold. Match B $\longleftrightarrow$ B' is certain, and matches A $\longleftrightarrow$ F and E $\longleftrightarrow$ E' are possible matches.

	A	B	C	D	E
B'	0.22	**0.92**	0.07	0	0
F	**0.60**	0.60	0.07	0	0
E'	0	0	0	0.58	**0.64**
G	0	0.07	0	**0.07**	0.02

Table 2. Accumulated similarity matrix M

Possible matches indicate that not enough duplicates

have been used. For instance, other duplicates whose S-tuple has differing values for B' and F would make B $\longleftrightarrow$ B' certain. Thus, when encountering possible matches, our algorithm iterates back to the duplicate detection step to find more duplicates. Note that the Whirl algorithm perfectly fits this requirement: After returning the first K duplicates, the current state of the search can be saved. When additional duplicates are required, the search can resume without having to recalculate the first K duplicates. If the set of certain and possible matches does not change in the second (or subsequent) run of the schema matching step, the algorithm terminates and both certain and possible matches are presented to the user.

When using this schema matching algorithm one has to determine a meaningful number for K. As we show in Section 6, the precision is equal or close to 100% at the early recall levels. Thus, we propose to use a small K to avoid false positives. The algorithm will automatically determine if the number was sufficient by computing the certainty of the resulting correspondences, and more duplicates will we detected by iterating back to the duplicate detection step.

6 Experiments

To show the feasibility and effectiveness of our approach, we performed experiments on real-world data. We also performed experiments in a controlled environment using synthetic data to answer two questions:

1. How effective is the duplicate detection algorithm in finding the top K duplicates?
2. How effective is the schema matching algorithm in finding a complete matching of two schemas, given K duplicates?

In answering the first question, we do not expect a better performance in finding *all* duplicates than other known domain-independent algorithms, because we do not assume aligned schemas. However, we do expect very good performance towards our goal of finding only the top K duplicates. In answering the second question, we expect to perform substantially better than known algorithms, because the schema matching algorithm has helpful input to work with, namely the duplicates.

All experiments were performed on a PC with an Athlon XP 1700+ CPU and 512 MB RAM running Windows 2000. The implementation is written in Java. We employ two well-known measures from the information retrieval field, which have also been used in schema matching evaluations [9]: precision and recall. These measures are defined as follows:

$$Precision = \frac{|D \cap R|}{|R|} \qquad Recall = \frac{|D \cap R|}{|D|}$$

where D is the set of existing duplicates, and R is the set of retrieved tuple pairs. In our duplicate detection experiments we only report the precision. The overall recall is not relevant because we are only interested in the top-K tuple pairs. However, both measures are important for the schema matching step: Low precision implies that false correspondences have to be manually deleted by the user, while low recall indicates that missing correspondences have to be manually added.

6.1 Experiments with apartment advertisements

The assumptions that our algorithm is based on are described in Section 1: We require data for both databases, and at least a few real-world objects must be represented in both databases. Examples of scenarios where duplicates actually occur were given there. In our experiments we used apartment advertisements extracted from web sites of two major newspapers in Berlin, Germany. Data had been extracted from each source on two consecutive weeks, i.e., altogether we could test our algorithm on four databases. We performed experiments with all six possible pairs of databases. Each database consisted of eleven attributes, each of which corresponded to another attribute in the other databases. The number of tuples ranged between 1509 and 3772.

Setting K to 10, we achieved 100% precision in all duplicate detection experiments. With that many very expressive duplicates, the schema matching process was also successful in detecting all existing correspondences. The reason for these excellent results are obvious: People tend to place their advertisements in more than one newspaper. Furthermore, the same or similar expressions are used in those duplicates. Thus, we are able to detect some very expressive duplicates, which makes schema matching straightforward.

In order to determine how our algorithm behaves in cases where only few duplicates exist or the intensional overlap is small, we conducted several experiments in a controlled environment using generated data. In particular, we address the problems described in Section 3.

6.2 Experimental setup

For a data source we used the data generator tool G2[2] used in [3], which improves the database generator used in [17] to create more realistic data. The tool allowed us to inject fuzzy duplicates, where fuzziness is controlled for each attribute through error probability parameters for different types of errors, such as “replacement error” and “deletion error”. For each experiment we generated two databases *DB1* and *DB2*, each with 5,000 tuples. Figure 4

[2]Kindly provided to us by Luca De Santis of the DaQuinCIS team at Universitá di Roma “La Sapienza”.

shows the schemas and the correct matching. Unbeknownst to our schema matching algorithm, the two databases have up to six attributes in common and each has two or more additional attributes. Attribute values were randomly chosen from long, predefined lists of values. Note that the attribute pairs Birth-place and City as well as Birth-district and District draw values from the same domains, making duplicate detection challenging. Finally, the columns were randomly shuffled. All reported results are averages of five independent runs with newly created databases for each experiment.

Attr. of *DB1*		Attr. of *DB2*
SSN	$\longleftrightarrow$	–
Profession	$\longleftrightarrow$	–
Surname	$\longleftrightarrow$	Surname
Name	$\longleftrightarrow$	Name
Birth-date	$\longleftrightarrow$	Birth-date
Birth-place	$\longleftrightarrow$	Birth-place
Birth-district	$\longleftrightarrow$	Birth-district
Sex	$\longleftrightarrow$	Sex
–	$\longleftrightarrow$	City
–	$\longleftrightarrow$	District

Figure 4. The correct matching from *DB1* to *DB2*

6.3 Duplicate Detection

To gauge how effective the *tupsim*(r, s) measure is in detecting duplicates without schema information, we performed recall/precision analyses, using standard interpolation.

Figure 5 shows the recall/precision diagram of our first experiment. For each curve we injected a different number of duplicates (10, 50, and 100) into the database. We observe that all results are favorable, i.e., the measure does a very good job of ranking duplicates higher than non-duplicates. Recall that we are interested only in the top K tuple pairs. The recall/precision diagram shows that the measure is extremely precise in particular for the top-ranked results. Consider the curve for 50 duplicates and regard the perfect precision of 100 percent at 50 percent recall. This means that in this experiment the first 25 tuple pairs returned by the duplicate detection algorithm were indeed duplicates.

Our next experiment was designed to assess the influence of intensional overlap, i.e., the number of common attributes in both databases. The schema of *DB1* remained the same. From the schema of *DB2* we successively removed the overlapping attributes Sex, Birth-district, and Birth-place, replacing them successively by new attributes Street number, Address, and Postal code. We used a

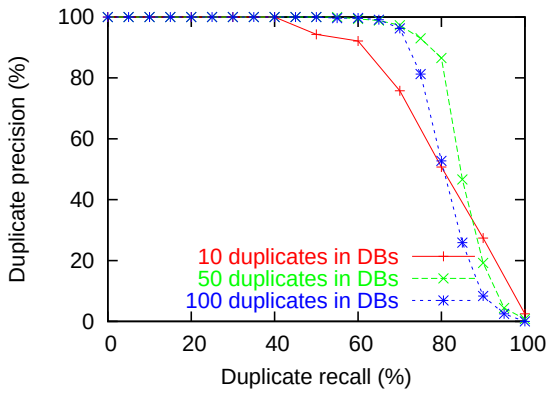

Figure 5. Influence of number of duplicates

fixed number of 50 duplicates. As expected, the results in Figure 6 show higher precision with increasing intensional overlap.

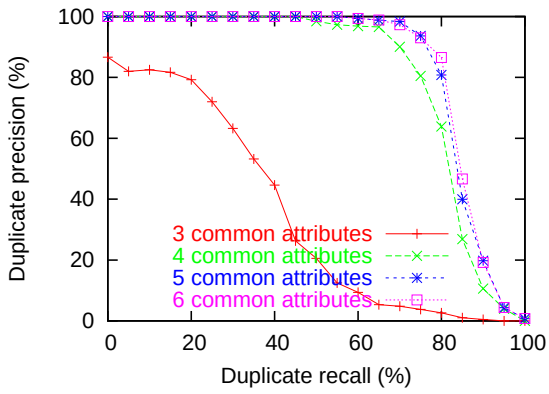

Figure 6. Influence of degree of intensional overlap

Most notable is the decrease in precision when only three attributes correspond, which calls for some explanation. In this experiment, both attributes Birth-place and Birth-district are not part of $DB2$. However, attributes City and District are still in $DB2$. Because these attributes draw values from the same domain, as described above, the precision is heavily decreased. Also note that a similar case is produced in the step from five to four common attributes, when Birth-district is removed from the schema of $DB2$. The decrease is not as drastic as in the previously mentioned case, because there is only a small number of districts. Hence, the values for District and Birth-District have a low inverse document frequency, and thus, only a small effect on the similarity *tupsim*.

To summarize, we have evaluated the effectiveness of our duplicate detection method, and thus, also addressed

the first three problems described in Section 3: Using a similarity measure we are able to detect fuzzy duplicates (first problem). Both extensional overlap (number of duplicates) and intensional overlap (number of corresponding attributes) affect the precision of duplicate detection, but it is still possible to detect a few duplicates even in low overlap situations (problems two and three). Apart from being effective, the algorithm has also shown good performance. Recall that we require only a few duplicates for schema matching, and these can be quickly detected with our adaptation of the Whirl algorithm: finding the top ten tuple pairs took less than one second on average. However, as the last experiment has shown, in some special cases it might be necessary to improve precision using prior knowledge, such as known attribute correspondences.

6.4 Duplicate detection using known correspondences

In some situations a few attributes are known to correspond, e.g., by using a simple name-based attribute matcher or by human input. We adapted our tuple similarity measure to incorporate such previous knowledge: Let $m_1, \ldots, m_k$ be the pairs of already matched attributes and let $r[m_i]$ and $s[m_i]$ denote the values of tuples r and s in the matched attribute of m_i, respectively. Finally, let r_u and s_u be the remaining, unmatched parts of tuples r and s. Our *extended tuple similarity measure* ($etupsim$) of two partially aligned tuples r and s is the average of the similarity of the unmatched part and the similarity scores of each matched attribute:

$$etupsim(r,s) := \frac{1}{k+1}\left(tupsim(r_u, s_u) + \sum_{i=1}^{k} fieldsim(r[m_i], s[m_i])\right) \quad (7)$$

By treating the set of matched attributes individually and the set of unmatched attributes as a whole, $etupsim(r,s)$ weighs matched attributes higher than unmatched ones. This preference reflects the intuition that duplicates should be similar in known corresponding attributes but must not necessarily be similar in the remaining, possibly not corresponding attributes. Note that $etupsim(r,s)$ is a generalization of $tupsim(r,s)$ of Section 4.2; if no attributes are matched then $etupsim(r,s) = tupsim(r,s)$.

We show that this measure results in improved precision in the duplicate detection procedure. For this experiment with 50 duplicates we reduced the intensional overlap shown in Figure 4 as follows: In *DB2* we replaced Birth-district and Sex with two new attributes Address and Street-number. Thus, only four attributes remain in common and each database had four other attributes. Figure 7 shows the results for different numbers of partial matches as input. As expected, knowledge of partial matches indeed improves duplicate detection.

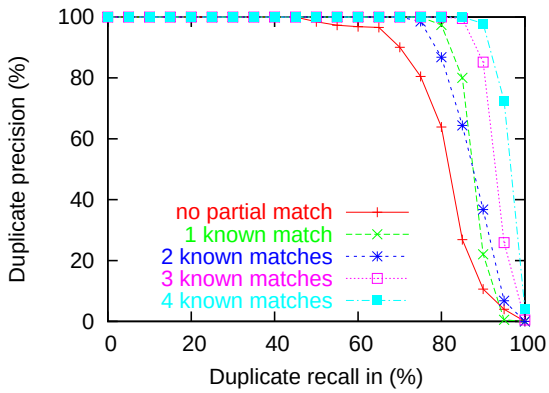

Figure 7. Influence of degree of partial alignment

6.5 Schema Matching

In this section we show how helpful duplicates are for performing schema matching. It is important to note that the main factor for good schema matching results is the ability to avoid using non-duplicates in the schema matching step, i.e., having no or only few false positives among the top-ranked results. This ability was demonstrated in the previous section. In the next paragraphs we show that usually few duplicates suffice to find at least some if not all correct matchings. We also report on our experience that a few false positives can usually be compensated and do not degrade the matching.

In Figure 8 we report on six experiments with the two databases *DB1* and *DB2* having four attributes in common. In the first experiment, we injected three duplicates and searched for the best two ($K = 2$) for schema matching. On average, we found 95 percent of the correct matching (recall), and of all matches that our algorithm suggested only 10 percent were incorrect (precision). The scores are not perfect because once in a while even among the best two duplicates there was a false positive.

Note that to test the limits of our approach we report on extreme cases with very few duplicates in the databases and very low intensional overlap. In general, we observe that choosing a low K is recommended and as K approaches the true number of duplicates, more and more false positives pollute the matching. Choosing a small K is not necessarily disadvantageous, because our schema matching procedure can detect if more duplicates are required, as described in Section 5. The next experiment highlights this effect but shows a certain robustness of our approach towards an increasing number of false positives.

From a database with 50 duplicates we handpicked the top ten duplicates and performed schema matching based

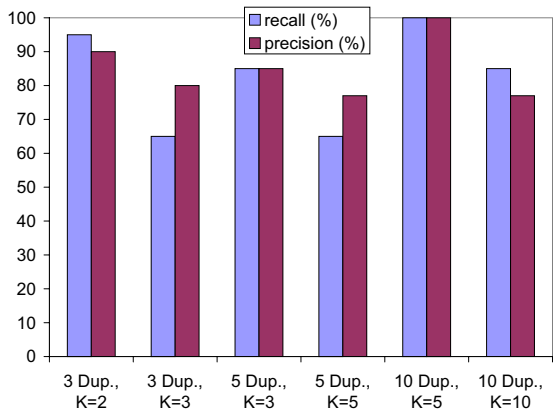

Figure 8. Precision and recall of schema matching

on this perfect choice. Next, we incrementally added the top 20 false positives, i.e., very similar tuple pairs that are known not to be duplicates. The results are presented in Figure 9, which shows that recall and precision of schema matching degrade only after as many false positives as true positives are used. The reason for this robustness is that all duplicates support a similar matching, while each false positive might support a different matching. Both curves level after a certain amount of noise (false positives) is introduced.

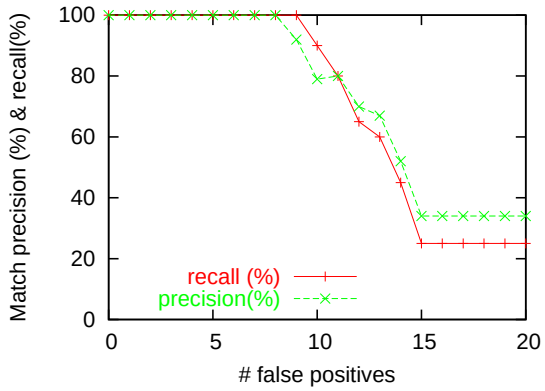

Figure 9. Robustness of schema matching with 10 true duplicates in the presence of false positives

By examining the robustness of our schema matching algorithm we also addressed the fourth problem of Section 3. Equal or highly similar values of attributes that do not correspond can mislead duplicate detection, and thus, induce a few false duplicates. However, as described in Section 6.3, the percentage of such false duplicates among the top-K tuple pairs is relatively low. Consequently, by using several tuple pairs (instead of using only the most similar one) we are still able to derive a correct schema matching.

So far we have assumed that schema matching immediately follows duplicate detection, and no human interaction is required. Consequently, in some situations too many false duplicates might be detected, resulting in poor schema matching performance. However, as shown in the previous experiments, the number of duplicates required for schema matching is not very large. Thus, the user can be included in the process by manually checking a few tuple pairs. Determining when to require human intervention is part of future research.

7 Conclusion and outlook

With this paper we have presented a novel approach to instance-based schema matching developed in the DUMAS project. Starting with the simple idea of using duplicates to detect corresponding attributes in tables, we identified the major and the subtle obstacles in achieving high-precision and high-recall schema matching. One major impediment is to find duplicates among tables with unmatched and only partially overlapping schemas. We have presented a similarity function that successfully identifies duplicates in such adverse settings, thereby making a contribution to the field of duplicate detection. In particular, using our duplication algorithm it is possible to find some duplicates even when the extensional overlap is small.

Next, we presented a procedure to generate a matching based on a small set of fuzzy duplicates, using alignment algorithms on similarity matrices. The effectiveness, robustness, and efficiency of our approach was shown in several experiments on artificial and real-world data. In particular, we were able to distinguish structurally similar but semantically different attributes, such as Birth-place and City. We also presented results on the helpfulness of partial matches to find duplicates using a combined similarity measure for matched and unmatched parts of the data.

We are currently working on techniques to improve efficiency also for this combined measure. Furthermore, we will develop methods for matching schemas consisting of multiple tables. Future work also includes investigation of two potential improvements: First, as in most related approaches (with the notable exception of [8]), we have not considered 1:n matches. Second, we also investigate the use of domain knowledge in the duplicate detection step.

Acknowledgements. We thank Mattis Neiling for providing apartment advertisement data. This research was supported by the Berlin-Brandenburg Graduate School on Distributed Information Systems (DFG grant GRK 316) and DFG grant NA 432.

References

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 586–597, 2002.

[2] P. A. Bernstein. Applying model management to classical meta data problems. In *First Biennial Conference on Innovative Database Research (CIDR)*, 2003.

[3] P. Bertolazzi, L. D. Santis, and M. Scannapieco. Automatic record matching in cooperative information systems. In *Proceedings of the International Workshop on Data Quality in Cooperative Information Systsems (DQCIS)*, Siena, Italy, 2003.

[4] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ACM International Conference on Knowledge discovery and data mining (SIGKDD)*, Washington, DC, 2003.

[5] C. E. H. Chua, R. H. L. Chiang, and E.-P. Lim. Instance-based attribute identification in database integration. *VLDB Journal*, 12(3):228–243, 2003.

[6] W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, Seattle, WA, 1998.

[7] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI Workshop on Information Integration on the Web (IIWeb)*, pages 73–78, Acapulco, Mexico, 2003.

[8] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering complex semantic matches between database schemas. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 383–394, 2004.

[9] H.-H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Web, Web-Services, and Database Systems*, Springer LNCS 2593, pages 221–237, 2002.

[10] H.-H. Do and E. Rahm. COMA - A system for flexible combination of schema matching approaches. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 610–621, Hong Kong, China, 2002.

[11] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 509–520, Santa Barbara, CA, 2001.

[12] A. Doan, Y. Lu, Y. Lee, and J. Han. Object matching for data integration: A profile-based approach. In *Proceedings of the IJCAI Workshop on Information Integration on the Web (IIWeb)*, Acapulco, Mexico, 2003.

[13] M. G. Elfeky, V. S. Verykios, and A. K. Elmagarmid. TAILOR: A record linkage toolbox. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 17–28, 2002.

[14] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[15] Z. Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys*, 18(1):23–38, 1986.

[16] L. Gravano, P. G. Ipeiriotis, N. Koudas, and D. Srivastava. Text joins in an RDBMS for web data integration. In *Twelfth International World Wide Web Conference*, pages 90–101, 2003.

[17] M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[18] E.-P. Lim, J. Srivastasa, S. Prabhakar, and J. Richardson. Entity identification in database integration. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 294–301, Vienna, Austria, 1993.

[19] D. Lopresti and A. Tomkins. Block edit models for approximate string matching. *Theoretical Computer Science*, 181:159–179, 1997.

[20] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Rome, Italy, 2001.

[21] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*, San Jose, CA, 2002.

[22] A. E. Monge and C. P. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 267–270, 1996.

[23] F. Naumann, C.-T. Ho, X. Tian, L. Haas, and N. Megiddo. Attribute classification using feature analysis. In *Proceedings of the International Conference on Data Engineering (ICDE)*, San Jose, CA, 2002. poster.

[24] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[25] M. Perkowitz and O. Etzioni. Category translation: Learning to understand information on the internet. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Montral, Canada, 1995.

[26] L. Popa, Y. Velegrakis, R. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 598–609, 2002.

[27] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[28] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[29] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

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

[31] W. E. Winkler. Matching and record linkage. In *Business Survey Methods*. Wiley-Interscience, 1995.