Database Schema Matching using Corpus-based Semantic Similarity and Word Segmentation

Aminul Islam, Diana Inkpen, and Iluju Kiringa

University of Ottawa, School of Information Technology and Engineering
800 King Edward, Ottawa, ON, K1N 6N5, Canada
{mdislam, diana, kiringa}@site.uottawa,ca

Abstract. In this paper, we present a new method for database schema matching, the problem of identifying elements of two given schemas that correspond to each other. We use two methods based on a large text corpus: one method for determining the semantic similarity of two target words and the other for automatic word segmentation. We present a name-based element-level database schema matching method that exploits the semantic similarity and the word segmentation method. We also use normalized and modified versions of the Longest Common Subsequence string matching algorithm with weight factors to allow for a balanced combination. Our goal is to develop a schema matching method that uses a single property (element name) for matching and achieves a comparable F-measure score with respect to the methods that use multiple properties (element name, text description, data instance, context description). We validate our method with experimental studies, the results of which suggest that the method is a useful addition to the set of existing schema matchers.

1 Introduction

Database schema matching is the problem of identifying elements of two given schemas that correspond to each other. It has been the focus of research since the 1970s in the Artificial Intelligence, Databases, and Knowledge Representation communities. Schema matching can also be defined as discovering semantically corresponding attributes in different schemas or detecting two names that denote the same concept in a flat ontology. Traditionally, the problem of matching schemas has essentially relied on finding pairwise attribute correspondences. Though schema matching identifies elements that correspond to each other, it does not explain how they correspond. For example, it might say that FirstName and LastName in one schema are related to Name in the other, but it does not say that concatenating the former yields the latter. Automatically discovering these correspondences or matches is inherently difficult.

Today, many researchers realize that schema matching is a core problem in e-commerce exchanges, in data integration / warehousing, and in Semantic Web applications. Schema matching is fundamental for enabling query mediation and data exchange across information sources [2], [21]. While schema matching has always been a problematic and interesting aspect of information integration, the problem is exacerbated as the number of information sources to be integrated, and hence the number of

integration problems that must be solved, grows. Such schema matching problems arise both in "classical" scenarios such as company mergers, and in "new" scenarios such as the integration of diverse sets of queryable information sources over the Web. Purely manual solutions to the schema matching problem are too labor-intensive to be scalable; as a result, there has been a great deal of research into automated techniques that speed up this process by either automatically-discovering good mappings, or at least by proposing likely matches that are then verified by a human expert [9].

Rahm and Bernstein [18] point out that it is not possible to determine fully-automatically all the matches between two schemas, primarily because most schemas have some semantics that affects the matching criteria but is not formally expressed or often not even documented. The implementation of the matching should therefore only determine match candidates, which the user can accept, reject, or change. Furthermore, the user should be able to specify matches for elements for which the system was unable to find satisfactory match candidates.

In this paper we present a novel approach to database schema matching, by using natural language processing techniques. The paper is organized as follow: Section 2 presents a short overview of different schema matching approaches. The corpus-based word similarity method and the word segmentation method that we use in schema matching are briefly described in Section 3. Our proposed schema matching method is described in Section 4 and examples are given in Section 5. Evaluation and experimental results are presented in Section 6 and we conclude in Section 7.

2 Classification of Schema Matching Approaches

Rahm and Bernstein [18] summarize the major approaches to schema matching. There are individual matchers: each computes a mapping based on a single matching criterion. Alternatively, combinations of individual matchers are built, either by using multiple matching criteria (e.g., name and type equality) within an integrated *hybrid matcher* or by combining multiple match results produced by different match algorithms within a *composite matcher*.

Among the individual matchers, linguistic matchers are of interest to us. They use element names and text (sentences) to find semantically similar schema elements. We discuss here two linguistic approaches: a) name matching and b) description matching.

a. Element Name Matching

Element name-based matching matches schema elements with equal or similar names. Similarity of names can be defined and measured in various ways, including:

- Equality of name matching
- Equality of canonical name representations after stemming and preprocessing
- Equality of synonyms
- Equality of hypernyms (words that are more general)
- Similarity of names based on longest common substrings (LCS), edit distance, pronunciation, soundex, or other string similarity measures.

Solving any task related to synonyms and hypernyms normally requires the use of thesauri or dictionaries. These specific dictionaries require a substantial effort to be built up in a consistent way. But corpus-based methods could be a better choice than

dictionary-based methods as a balanced text corpus covers a huge collection of both domain-dependent and independent words including special terms and proper nouns.

Name-based matching can identify multiple relevant matches for a given schema element i.e., it is not limited to finding just 1:1 matches. For example, it can match "address" with both "home address" and "office address". Bright et. al. [3] discuss an approach to assigning different weights to different types of similarity relations.

b. Description matching

Often, schemas contain text descriptions of elements that typically explain the meaning of elements in natural language to express the intended semantics of schema elements. But the quality of these descriptions varies a lot. These comments can also be evaluated linguistically to determine the similarity between schema elements. For instance, this would help find that the following elements match, by a linguistic analysis of the comments associated with each schema element:

S1: empn // employee name

S2: name // name of employee

This linguistic analysis could be as simple as extracting keywords from the description which are used for synonym comparison, much like name matching. Some approaches consider rule-based schema matching which are domain dependent [16].

Madhavan et al. [13] use name matching and description matching as part of a combined method that builds a model for each schema element that includes knowledge about other elements in a corpus of schemas and uses this model in the matching process. Specifically, given the element in a schema that is not in the corpus, it finds other elements in the corpus that are an alternate representation of the same underlying concept. The method uses the corpus of schemas to estimate various statistics about elements and relations in a domain to develop a better understanding of the domain. They use 4 base learners (name learner, text learner, data instance learner, and context learner) and a meta learner. For example, the name learner first tries to identify frequent word roots in the element names by first splitting the names of the elements based on capitalization and stemming the resulting fragments. Then it splits the names into their corresponding n-grams to handle short forms, incomplete names and spelling errors that are common in schema names. Finally, the method uses each base learner to make a prediction of how a schema element is similar to each of the corpus elements. It combines the predictions of the base learners into a single similarity score.

3 Two corpus-based methods

We were motivated to use corpus-based similarity and word segmentation methods for the following reasons (by corpus here we mean a large collection of text). First, we focused our attention on corpus-based measures because of their large type coverage. The types that are used in real-world database schema elements are often not found in dictionaries. Second, some existing corpus-based word segmentation methods provide good precision score, but provide low recall, and as a result low F-measure score.

3.1 Word Similarity Method

There is a relatively large number of word-to-word similarity metrics in the literature, ranging from distance-oriented measures computed on semantic networks or knowledge base (or dictionary / thesaurus-based measures), to metrics based on models of information theory (or corpus-based measures) learned from large text collections. A detailed review on word similarity can be found in [19], [23]. We choose a corpus-based similarity measure because of the large type coverage.

PMI-IR [22] is a simple method for computing corpus-based similarity of words. It uses Pointwise Mutual Information, PMI(w_1, w_2) = log p(w_1 & w_2) / p(w_1) p(w_2). Here, w_1 and w_2 are the two words; p(w_1 & w_2) is the probability that the two words co-occur. If w_1 and w_2 are statistically independent, then the probability that they co-occur is given by the product p(w_1) · p(w_2). If they are not independent, and they have a tendency to co-occur, then p(w_1 & w_2) will be greater than p(w_1) · p(w_2). PMI-IR used AltaVista Advanced Search query syntax to calculate the probabilities. In the simplest case, two words co-occur when they appear in the same document. The probabilities can be approximated by the number of documents (hits) retrieved:

PMI-IR(w_1, w_2) = hits(w_1 AND w_2) / (hits(w_1) hits(w_2)).

Latent Semantic Analysis (LSA) [12], a high-dimensional linear association model, analyzes a large corpus of natural text and generate a representation that captures the similarity of words and text passages. The underlying idea is that the aggregation of all the word contexts in which a given word does and does not appear provides a set of mutual constraints that largely determines the similarity of meaning of words and sets of words to each other [12]. The model tries to answer how people acquire as much knowledge as they do on the basis of as little information as they get. It uses the Singular Value Decomposition (SVD) to find the semantic representations of words by analyzing the statistical relationships among words in a large corpus of text. The similarity of two words is measured by the cosine of the angle between their corresponding vectors.

We use **Second Order Co-occurrence PMI** (SOC-PMI) word similarity method [7] that uses Pointwise Mutual Information to sort lists of important neighbor words of the two target words from a large corpus. The method considers the words which are common in both lists and aggregate their PMI values (from the opposite list) to calculate the relative semantic similarity. We empirically evaluated this method [7] by computing its correlation with the human scores for the Miller and Charles's [15] 30 noun pair subset and the Rubenstein and Goodenough's [20] 65 noun pairs. The evaluation also included the use of the word similarity method in the task of solving 80 synonym test questions from the Test of English as a Foreign Language (TOEFL), and 50 synonym test questions from a collection of English as a Second Language (ESL) tests. The evaluation results show that the method outperforms several competing methods (PMI-IR and LSA).

PMI-IR used AltaVista Advanced Search query syntax to calculate the probabilities. The 'NEAR' search operator of AltaVista is an essential operator in PMI-IR method and it is no longer in use in AltaVista; this means that it is practically not possible to use PMI-IR method in the same form in new systems. Also, we prefer to SOC-PMI because it uses second-order co-occurrences, therefore it can compute similarity even for two words that do not co-occur in the corpus. The word similarity method is a separate module in our Schema Matching Method. Therefore any other

word similarity method could be substituted instead of SOC-PMI, if someone wants to try other word-similarity methods (dictionary-based, corpus-based, or hybrid).

3.2 Word Segmentation Model

Word segmentation methods can be roughly classified as either dictionary-based or corpus-based methods, while many state-of-the-art systems use hybrid approaches. In dictionary-based methods, given an input character string, only words that are stored in the dictionary can be identified. The performance of these methods thus depends to a large degree upon the coverage of the dictionary, which unfortunately may never be complete because new words appear constantly. Therefore, in addition to the dictionary, many systems contain special components for unknown word identification. In particular, statistical corpus-based methods have been widely applied because they use a probabilistic scoring mechanism rather than a dictionary to segment the text [6].

We use a corpus-based method for automatic word segmentation [8]. The method formulates a generalized approach to word segmentation using maximum-length descending-frequency and entropy rate. The term *maximum-length descending-frequency* means that it chooses maximum length *n*-grams (sequences of *n* characters) that have a minimum threshold frequency; then it looks for further *n*-grams in descending order, based on length. If two *n*-grams have the same length, it chooses the *n*-gram with highest frequency first and then the *n*-gram with next-highest frequency if any of its characters are not a part of the previous one. Following this procedure, after some iterations, it can be in a state with some remaining characters (they call it *residue*) that is not matched with any type in the corpus. To solve this, the method merges *residue* with its adjacent words to form a string of characters and then apply a greedy matching from the beginning and the end of the string; this is an algorithm of *forward-backward matching* type [4], in which the results are composed and the segmentation optimized based on the two results. The method chooses the result with lower number of words. If the two results return same number of words then it uses the entropy rate to decide which set of words to accept. The intuition behind using entropy rate is that if it has a set of words with larger average frequency (normalized frequency in the entropy rate) than the other set of words, it is obvious that the first set of words is more meaningful than the second set of words [8]. The method obtained 89.92% word precision rate, 94.69% word recall rate, and 92.24% word F-measure when they tested the segmentation method on the Brown corpus. The results of other word segmentation methods ([5], [17], [10]), which are also tested on the Brown corpus, show that our method outperforms these methods in terms of precision, recall, and F-measure.

4. Proposed Schema Matching Method

We use the *longest common subsequence* (LCS) [1] measure with some normalization and small modifications for our string similarity measure. We use three different modified versions of LCS and then take a weighted sum of these[1]. Kondrak [11]

[1] We use modified versions because in our experiments we obtained better results (precision and recall for schema matching on a sample of data) than when using the original LCS, or other similarity measures.

showed that edit distance and the length of the longest common subsequence are special cases of *n*-gram distance and similarity, respectively. Melamed [14] normalized LCS by dividing the length of the longest common subsequence by the length of the longer string and called it *longest common subsequence ratio* (LCSR). But LCSR does not take into account of the length of the smaller string which sometimes has a significant impact on the similarity score.

We normalize the *longest common subsequence* (LCS) so that it takes into account of the length of both the smaller and the longer string and call it *normalized longest common subsequence* (NLCS) which is,

$$v_1 = NLCS\ (r_i, s_j) = \frac{\{length\ (LCS\ (r_i, s_j))\}^2}{length\ (r_i) \times length\ (s_j)}$$

While in classical LCS, the common subsequence need not to be consecutive, in database schema matching, consecutive common subsequence is important for a high degree of matching. We use *maximal consecutive longest common subsequence* starting at character 1, *MCLCS1* (Figure 1) and *maximal consecutive longest common subsequence* starting at any character *n*, *MCLCSN* (Figure 2). In Figure 1, we present an algorithm that takes two strings as input and returns the smaller string or maximal consecutive portions of the smaller string that consecutively match with the longer string, where matching must be from first character (character 1) for both strings. In Figure 2, we present another algorithm that takes two strings as input and returns the smaller string or maximal consecutive portions of the smaller string that consecutively match with the longer string, where matching may start from any character (character *n*) for both of the strings. We also normalize *MCLCS1* and *MCLCSN* and call it *normalized MCLCS1* (v_2) and *normalized MCLCSN* (v_3), respectively.

We take the weighted sum of these individual v_1, v_2, and v_3 to determine string similarity score, where w_1, w_2, w_3 are weights and $w_1+w_2+w_3=1$. Therefore, the similarity of the two strings is: $\alpha = w_1v_1 + w_2v_2 + w_3v_3$. We set equal weights for our experiments. Theoretically, $v_3 \geq v_2$. We then use the word similarity measure, normalize it, and combine it with the string similarity to obtain the final similarity score.

We now describe our schema matching method in detail. Consider two given database schemas $R = \{R_1, R_2 \ldots, R_\sigma\}$ and $S = \{S_1, S_2 \ldots, S_\chi\}$; for each element in one database schema, we try to identify a matching element in the other schema, if any, using element names. We assume that schema R has σ elements and R_i is the element's name, where $i = 1 \ldots \sigma$. Similarly, schema S has χ elements and S_j is the element's name where $j = 1 \ldots \chi$. Note that some elements in R can match multiple elements in S, and vice versa. So, our task is to identify whether an element name $R_i \in R$ matches an element name $S_j \in S$. Both R_i and S_j are strings of characters. Our method provides a similarity score between 0 and 1, inclusively. If the similarity score is above a certain threshold then the elements are considered *match candidates*. If we set the threshold to 1 and the similarity score reaches this value, only then are we certain about their matching. For all other cases, we can only determine more or less probable *match candidates*. The method comprises the following six steps:

Step 1: We use all special characters, punctuations, and capital letters, if any, as initial word boundary and eliminate all these special characters and punctuations. After this

initial word segmentation, we pass the segmented words to the word segmentation method and lemmatize to generate tokens. We assume $R_i = \{r_1, r_2 \ldots, r_m\}$ has m tokens and $S_j = \{s_1, s_2 \ldots, s_n\}$ has n tokens and $n \geq m$. Otherwise, we switch R_i and S_j.

Step 2: We count the number of r_i's (say, δ) for which $r_i = s_j$, for all $r \in R_i$ and for all $s \in S_i$. I.e., there are δ tokens in R_i that exactly match with S_j, where $\delta \leq m$. We remove all δ tokens from both of R_i and S_j. So, $R_i = \{r_1, r_2 \ldots, r_{m-\delta}\}$ and $S_j = \{s_1, s_2 \ldots, s_{n-\delta}\}$. If $m-\delta = 0$, we go to step 6.

Step 3: We construct a $(m-\delta)\times(n-\delta)$ *matching matrix* (say, $M_1 = (\alpha_{ij})_{(m-\delta)\times(n-\delta)}$) using the following process: we assume any token $r_i \in R_i$ has τ characters, i.e., $r_i = \{c_1c_2\ldots c_\tau\}$ and any token $s_j \in S_j$ has η characters, i.e., $s_j = \{c_1c_2 \ldots c_\eta\}$ where $\tau \leq \eta$. In other words, η is the length of the longer token and τ is the length of the smaller token. We calculate the followings: $v_1 \leftarrow NLCS(r_i, s_j)$ $v_2 \leftarrow NMCLCS1(r_i, s_j)$ $v_3 \leftarrow NMCLCSN(r_i, s_j)$ $\alpha_{ij} \leftarrow w_1v_1 + w_2v_2 + w_3v_3$ i.e., α_{ij} is a weighted sum of v_1, v_2, and v_3 (equal weights). We put α_{ij} in row i and column j position of a matrix M_1 for all $i = 1..m-\delta$ and $j = 1..n-\delta$.

Step 4: We construct a $(m-\delta)\times(n-\delta)$ *similarity matrix* (say, $M_2 = (\beta_{ij})_{(m-\delta)\times(n-\delta)}$) using the following process: We put β_{ij} (the SOC-PMI similarity score) in row i and column j position of a matrix M_2 for all $i = 1 \ldots m-\delta$ and $j = 1 \ldots n-\delta$.

Step 5: We construct another $(m-\delta)\times(n-\delta)$ *joint matrix* (say, $M = (\gamma_{ij})_{(m-\delta)\times(n-\delta)}$) using $M \leftarrow \psi M_1 + \varphi M_2$ (i.e., $\gamma_{ij} = \psi\alpha_{ij} + \varphi\beta_{ij}$) where ψ is the *matching matrix* weight factor. φ is the *similarity matrix* weight factor, and $\psi + \varphi = 1$. Setting any one of these factors to 0 means that we do not include that matrix. Setting both of the factors to 0.5 means we consider them equally important.

After constructing the *joint matrix*, M, we find out the maximum-valued matrix-element, γ_{ij}. We add this matrix element to a list (say, ρ and $\rho \leftarrow \rho \cup \gamma_{ij}$) if $\gamma_{ij} \geq \varsigma$ (we will discuss about the similarity threshold, ς in next section). We remove all the matrix elements of i'th row and j'th column from M. We repeat the finding of the maximum-valued matrix-element, γ_{ij} adding it to ρ and removing all the matrix elements of the corresponding row and column until either $\gamma_{ij} < \varsigma$, or $m-\delta-|\rho| = 0$, or both.

Step 6: We sum up all the elements in ρ and add δ to it to get a total score. We multiply this total score by the reciprocal harmonic mean of m and n to obtain a balance similarity score between 0 and 1, inclusively.

$$Similarity\ \ Score\ (R_i, S_j) = \frac{(\delta + \sum_{i=1}^{|\rho|} \rho_i) \times (m + n)}{2mn}$$

Choosing the values of ζ and ς

ζ is the minimum number of characters for which we continue the matching process. Theoretically ζ could be any value between 1 and m inclusively. We set ζ to 1. If we set ζ to 1 then we can get expected matching result for small-length tokens. E.g., if we have three sample tokens named *min*, *max* and *similarity* and we set ζ to 1. The pair *min max* returns *m* and the pair *min similarity* returns Ø when we use *MCLCS1*. When we use *MCLCSN*, the first pair returns *m* and the second pair returns *mi*. But if we set ζ to 2, the pair *min max* returns Ø for both *MCLCS1* and *MCLCSN*. If we set ζ to 3, the pair *min similarity* returns Ø for both *MCLCS1* and *MCLCSN*.

Theoretically, ς could be any value between 0 and 1, but we usually set ς close to 0 (we set $\varsigma = 0.01$ for all of our experiments). All matrix elements having values lower than ς may have negative impacts to the matching, thus it is better to omit them.

Algorithm *MCLCS1*

Input: r_i, s_j // r_i and s_j are two input strings where $|r_i| = \tau$, $|s_j| = \eta$ and $\tau \leq \eta$ as mentioned.

1. $\tau \leftarrow |r_i|$, $\eta \leftarrow |s_j|$
2. **while** $|r_i| \geq \zeta$ // we usually set ζ to 1. Details are discussed in next section.
3. **if** $r_i \in S_j$ // i.e., $S_j \bigcap r_i = r_i$
4. **return** r_i
5. **else** $r_i \leftarrow r_i \setminus c_\tau$ // i.e., remove the right-most character from r_i
6. **end if**
7. **end while**

Output: r_i // r_i is the Maximal Consecutive LCS starting at character 1

Figure 1. Maximal Consecutive LCS starting at character 1.

Algorithm *MCLCSN*

Input: r_i, s_j // r_i and s_j are two input strings where $|r_i| = \tau$, $|s_j| = \eta$ and $\tau \leq \eta$.

1. **while** $|r_i| \geq \zeta$ // we usually set ζ to 1.
2. determine all n-grams from r_i where $n = 1 .. |r_i|$ and $\overline{r_i}$ is the set of n-grams
3. **if** $x \in s_j$ where $\{x \mid x \in \overline{r_i}, x = \mathbf{Max}(\overline{r_i})\}$

// i is the number of n-grams and Max ($\overline{r_i}$) returns the maximum length n-gram from $\overline{r_i}$

4. **return** x
5. **else** $\overline{r_i} \leftarrow \overline{r_i} \setminus x$ // remove x from set $\overline{r_i}$
6. **end if**
7. **end while**

Output: x // x is the Maximal Consecutive LCS starting at any character n

Figure 2. Maximal consecutive LCS starting at any character *n*

5. Example

We provide an example that describes the proposed method and determine the similarity score. We use two simple element names from a database schema, for brevity.

Let R_i = "maxprice", S_j = "High_Price".

Step 1: After eliminating all special characters and punctuations, if any, and then using word segmentation method and lemmatizing , we get R_i = {*max, price*} and S_j = {*high*, *price*} where $m = 2$ and $n = 2$.

Step 2: Because only one token (i.e., *price*) in R_i exactly matches with S_j we set δ to 1. We remove *price* from both R_i and S_j. So, R_i = {*max*} and S_j = {*high*}. As $m - \delta \neq 0$, we proceed to next step.

Step 3: We construct a 1×1 *matching matrix*, M_1. Consider the *max high* pair where $\eta = 4$ is the length of the longer token (*high*), $\tau = 3$ is the length of the smaller

token (*max*) and 0 is the maximal length of the consecutive portions of the smaller token that consecutively match with the longer token. So, $v_1 = v_2 = v_3 = 0$ and $\alpha_{11} = 0$.

$$M_1 = \begin{matrix} \\ max \end{matrix} \begin{matrix} high \\ \begin{bmatrix} 0 \end{bmatrix} \end{matrix}$$

Step 4: We construct a 1×1 *similarity matrix*, M_2. Here, $\lambda = 20$ as we used the *SOCPMI* method.

$$M_2 = \begin{matrix} \\ max \end{matrix} \begin{matrix} high \\ \begin{bmatrix} 0.326 \end{bmatrix} \end{matrix}$$

Step 5: We construct a 1×1 *joint matrix*, M and assign equal weight factor by setting both ψ and φ to 0.5.

$$M = \begin{matrix} \\ max \end{matrix} \begin{matrix} high \\ \begin{bmatrix} 0.163 \end{bmatrix} \end{matrix}$$

We find the only maximum-valued-matrix-element, $\gamma_{ij} = 0.163$ and add it to ρ as $\gamma_{ij} \geq \varsigma$ (we use $\varsigma = 0.01$ in this example). So, $\rho = \{0.163\}$. The new M is empty after removing i'th ($i = 1$) row and j'th ($j = 1$) column. We proceed to next step as m-δ-$|\rho| = 0$. (Here, $m = 2$, $\delta = 1$ and $|\rho| = 1$.)

Step 6: $SimilarityScore(R_i, S_j) = \dfrac{(\delta + \sum_{i=1}^{|\rho|} \rho_i) \times (m+n)}{2mn} = (1 + 0.163) \times 4/8 = 0.582$

6. Evaluation and Results

We now present experimental results that demonstrate the performance of our method. All the schemas we used in our experiments are from Madhavan et al. [13], where they used web form schemas from two different domains, *auto* and *real estate*. Web form schema matching is the problem of identifying corresponding input fields in the web forms. Each web form schema is a set of elements, one for each input. The properties of each input include: the hidden input name or element name that is passed to the server when the form is processed, the description text and sample values in the option box. We tested on the same data as Madhavan et al. [13], all of it, while they used 75% of it, randomly selected. We could not reproduce the exact 75% that they used.

In each domain, they manually created mappings between randomly chosen schema pairs. The matches were *one-many*, i.e., an element can match any number of elements in the other schema. These manually created mappings are used as a *gold standard* to compare the mapping performance of the different methods, including our method. Table 1 provides detailed information about each of the two domains and our results.

For each domain, we compared each predicted mapping pair against the manually created mapping pair. For our experiment, we only used element names for matching. We used eleven different similarity thresholds ranging from 0 to 1 with interval 0.1. For example, using the *auto* domain when we used similarity threshold 0.1, our method matched 961 elements, out of which 628 elements were among the 769 manually matched elements. The last three columns of the table show the precision, recall, and F-measure for the two domains, for the various threshold values. A low similarity threshold (≈ 0.2) gives the best F-measure score.

The reason for a lower similarity threshold to obtain a better F-measure score is that we always take into accounts both the string similarity and the semantic word similarity measures. If two strings have perfect semantic word similarity score (i.e. ≈ 1) and no string similarity score (i.e. ≈ 0), which is practically a perfect matching (e.g., car and vehicle), the total similarity score will be lower. Again, we multiply this total score by the reciprocal harmonic mean of m and n to obtain a balanced similarity score; this also lowers the final similarity value. When we use string similarity threshold score of 1 (i.e., matching the element names exactly, therefore no semantic similarity matching is included), we obtain recall values of 0.133 and 0.107 for the auto and real estate domains, respectively. We can consider these as baselines.

Table 1. Characteristics of the evaluation domains and our results.

Domain	No. of schemas	No. of manual mappings	No. of manually created mapping pairs	Similarity threshold score in our method	No. of predicted mapping pairs	No. of correct mapping pairs	Precision	Recall	F-measure
Auto	30	95	769	0	33116	769	0.02	1.00	0.05
				0.1	961	628	0.65	0.82	0.73
				0.2	769	596	0.78	0.78	0.78
				0.3	701	564	0.80	0.73	0.77
				0.4	689	558	0.81	0.73	0.77
				0.5	642	530	0.83	0.69	0.75
				0.6	501	424	0.85	0.55	0.67
				0.7	438	382	0.87	0.50	0.63
				0.8	200	192	0.96	0.25	0.40
				0.9	176	176	1.00	0.23	0.37
				1.0	103	103	1.00	0.13	0.24
Real estate	20	57	280	0	4262	280	0.07	1.00	0.12
				0.1	364	232	0.64	0.83	0.72
				0.2	310	211	0.68	0.75	0.72
				0.3	248	176	0.71	0.63	0.67
				0.4	228	173	0.76	0.62	0.68
				0.5	203	164	0.81	0.59	0.68
				0.6	155	130	0.84	0.46	0.60
				0.7	124	105	0.85	0.38	0.52
				0.8	59	55	0.93	0.20	0.32
				0.9	48	48	1.00	0.17	0.29
				1.0	30	30	1.00	0.11	0.19

Madhavan et al. [13] used three methods: direct, pivot and augment. They selected a random 25% of the manually created mappings in each domain as training data and tested on the remaining 75% of the mappings. In the augment method, they used different base learners such as name learner, text learner, data instance learner, context learner and then used a meta-learner to combine the predictions of the different base learners into a single similarity score. To train a learner, the augment method requires learner specific positive and negative examples for the element on which it is being

trained. The direct method uses the same base learners, but the training data for these learners is extracted only from the schemas being matched. Pivot is the method that computes cosine distance of the interpretation vectors of the two elements directly.

For the auto domain, the direct, pivot and augment methods achieved precision of around 0.76, 0.74 and 0.92, recall of around 0.74, 0.78, 0.72 and F-measure of around 0.73, 0.74 and 0.78 respectively. We achieved around 0.78 as precision, recall and F-measure with 0.2 as similarity threshold. For the real estate domain, the direct, pivot and augment methods achieved precision of 0.78, 0.71 and 0.76, recall 0.69, 0.74, 0.81 and F-measure of 0.71, 0.71 and 0.78, respectively. We achieved precision of 0.68, recall of 0.75, and F-measure of 0.72, with the same threshold.

Generally, it seems that precision matters more than recall in the schema matching problem. But pragmatically it is not possible to determine fully-automatically all matches between two schemas, and the implementation of the matching therefore only determine match candidates that are then verified by a human expert. If a human expert is involved in the verification procedure then recall is as important as precision.

Note that our algorithms assumed that most element names are tokenizable, but not all of them. There are indeed types of data where it was nearly impossible to obtain matches using element name matching. For such cases, we got very low similarity values. However, even by considering cases like this one, we obtained good results on our experimental data sets, which is from real-world web data sources. This means that this type of data is not very frequent in real-world web data sources.

6. Conclusions

Our schema matching method uses a single property (i.e., element name) for matching and achieves a comparable F-measure score with respect to the methods that use multiple properties (e.g., element name, text description, data instance, context description). If we use a single property instead of multiple properties, it can speed up the matching process which is important when schema matching is used in Peer-to-Peer (P2P) data management or online query processing in P2P environments. Our method is scalable, in the sense that, if needed, we could also add other properties (i.e., text description and context description) to obtain a better schema matching result. To deal with non-tokenizable cases, we also plan to combine our name-based schema matcher with other existing matchers, in order to address specific situations that our method does not cover. When the element names are not words or fragments of words, then we need to use an instance matcher that looks at the type of the values in two columns, or at the values of the instances. If the instances are words, we can re-use our semantic and string similarity matching at the level of the instances. Sometimes two columns might match if similar words are used to denote different fields in two different databases. In such cases, the precision of the matching can be increased by matching the text descriptions of the columns, when available. A word-level similarity measure can be used to determine the similarity level of two description texts.

References

1. Allison, L., and Dix, T. I. A Bit-String Longest-Common-Subsequence Algorithm. In Information Processing Letters. 23, (1986), 305-310.
2. Batini, C., Lenzerini, M., and Navathe, S. B. A comparative analysis of methodologies for database schema integration. ACM Computing Surveys, 18, 4, (1986), 323–364.
3. Bright, M. W., Hurson, A. R., and Pakzad, S. H. Automated resolution of semantic heterogeneity in multi databases. In Trans on Database Systems (TODS), 19, 2, (1994), 212–253.
4. Dale, R., Moisl, H., and Somers, H. Handbook of Natural Language Processing. Marcel Dekker, Inc. New York, (2000), 22-26.
5. de Marcken, C. The Unsupervised Acquisition of a Lexicon from Continuous Speech. Technical Report AI Memo No. 1558, M.I.T., Cambridge, MA, (1995).
6. Gao, J., Li, M., Wu, A. and Huang, C.-N. Chinese word segmentation and named entity recognition: a pragmatic approach. Computational Linguistics, 31, 4, (2005).
7. Islam, A., and Inkpen, D. Second Order Co-occurrence PMI for Determining the Semantic Similarity of Words. In Proceedings of the International Conference on Language Resources and Evaluation, Genoa, Italy, (2006).
8. Islam, A., Inkpen, D., and Kiringa, I. A Generalized Approach to Word Segmentation using Maximum Length Descending Frequency and Entropy Rate. In Procs. of the 8th Intl. Conf. on Intelligent Text Processing and Comp. Linguistics (CICLing 2007), (2007), 175-185.
9. Kang, J., and Naughton, J. F. On Schema Matching with Opaque Column Names and Data Values. In Proceedings of SIGMOD 2003, San Diego, CA, (2003).
10. Kit, C., and Wilks, Y. Unsupervised Learning of Word Boundary with Description Length Gain. In Proceedings CoNLL99 ACL Workshop. Bergen, (1999).
11. Kondrak, G. N-gram similarity and distance. In Procs of the 12h Intel. Conf. on String Processing and Information Retrieval, Buenos Aires, Argentina, (2005), 115-126.
12. Landauer, T. K., Foltz, P. W., and Laham, D. Introduction to Latent Semantic Analysis. Discourse Processes, 25, 2-3, (1998), 259-284.
13. Madhavan, J., Bernstein, P., Doan, A., and Halevy, A. Corpus-based Schema Matching. In Proceedings of the International Conference on Data Engineering (ICDE-05), (2005).
14. Melamed, I. D. Bitext maps and alignment via pattern recognition. Computational Linguistics, 25, 1, (1999), 107–130.
15. Miller, G. A., and Charles, W. G. Contextual correlates of semantic similarity. Language and Cognitive Processes, 6, 1, (1991), 1-28.
16. Milo, T., and Zohar, S. Using Schema Matching to Simplify Heterogeneous Data Translation. In Procs. of the Intl. Conf. on Very Large Data Bases (VLDB), (1998), 122-133.
17. Peng, F., and Schuurmans, D. A Hierarchical EM Approach to Word Segmentation, In Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium (NLPRS 2001), Tokyo, Japan, (2001), 475-480.
18. Rahm, E., and Bernstein, P.A. A survey of approaches to automatic schema matching. The International Journal on Very Large Data Bases (VLDB), 10, 4, (2001), 334-350.
19. Rodriguez, M.A., Egenhofer, M.J. Determining Semantic Similarity among Entity Classes from Different Ontologies. IEEE Trans. Knowledge and Data Eng., 15, 2, (2003), 442-456.
20. Rubenstein, H., and Goodenough, J. B. Contextual correlates of synonymy. Communications of the ACM, 8, 10, (1965), 627-633.
21. Seligman, L., Rosenthal, A., Lehner, P., and Smith, A. Data integration: Where does the time go? Bulletin of the Technical Committee on Data Engineering, 25, 3, (2002).
22. Turney, P. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In Proceedings of Twelfth European Conf. Machine Learning, (2001).
23. Weeds, J., Weir, D., and McCarthy, D. Characterising Measures of Lexical Distributional Similarity. Procs. of the 20th Intl. Conf. on Computational Linguistics, (2004), 1015-1021.