The VLDB Journal
DOI 10.1007/s00778-007-0067-9

REGULAR PAPER

Applications of corpus-based semantic similarity and word segmentation to database schema matching

Aminul Islam · Diana Inkpen · Iluju Kiringa

Received: 24 July 2006 / Revised: 8 May 2007 / Accepted: 8 July 2007

Abstract In this paper, we present a method for database schema matching: the problem of identifying elements of two given schemas that correspond to each other. Schema matching is useful in e-commerce exchanges, in data integration/warehousing, and in semantic web applications. We first present two corpus-based methods: one method is for determining the semantic similarity of two target words and the other is for automatic word segmentation. Then we present a name-based element-level database schema matching method that exploits both the semantic similarity and the word segmentation methods. Our word similarity method uses pointwise mutual information (PMI) to sort lists of important neighbor words of two target words; the words which are common in both lists are selected and their PMI values are aggregated to calculate the relative similarity score. Our word segmentation method uses corpus type frequency information to choose the type with maximum length and frequency from "desegmented" text. It also uses a modified forward–backward matching technique using maximum length frequency and entropy rate if any non-matching portions of the text exist. Finally, we exploit both the semantic similarity and the word segmentation methods in our proposed name-based element-level schema matching method. This method uses a single property (i.e., element name) for schema matching and nevertheless achieves a measure score that is comparable to the methods that use multiple properties (e.g., element name, text description, data instance, context description). Our schema matching method also uses normalized and modified versions of the longest common subsequence string matching algorithm with weight factors to allow for a balanced combination. We validate our methods with experimental studies, the results of which suggest that these methods can be a useful addition to the set of existing methods.

Keywords Database schema matching · Semantic similarity · Word segmentation · Corpus-based methods

A. Islam (✉) · D. Inkpen · I. Kiringa
School of Information Technology and Engineering,
University of Ottawa, Ottawa, Canada
e-mail: mdislam@site.uottawa.ca
URL: www.site.uottawa.ca/∼mdislam

D. Inkpen
e-mail: diana@site.uottawa.ca
URL: www.site.uottawa.ca/∼diana

I. Kiringa
e-mail: kiringa@site.uottawa.ca
URL: www.site.uottawa.ca/∼kiringa

1 Introduction

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. 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 [2,57]. 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 information sources queriable over the web.

We present a schema matching method that uses a single property (i.e., element name) for matching and achieves a comparable F-measure score compared to 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; this is important when schema matching is used in peer-to-peer (P2P) data management systems or in online query processing environments. If the properties that we use for schema matching contain element names or any types of text description, then we need to focus on both string matching and semantic similarity of words because sometimes only string matching or only semantic similarity of words provides good mapping results, and sometimes we need to use both in a balanced way. Names in schemas are often not segmented (words are connected together to form a name); therefore a good word segmentation method is required for better schema matching results. We propose two corpus-based methods: one for determining the semantic similarity of words and the other for word segmentation; then we formulate a name-based schema matching method that uses these two corpus-based methods. By corpus we mean a large collection of general-purpose English text.

We were motivated to propose corpus-based similarity and word segmentation methods for several reasons. First, we focused our attention on corpus-based measures because of their large type coverage (types of words). The types that are used in real-world database schema elements are often not found in dictionaries. Second, off-the-shelf corpus-based similarity measures are not as comparable as dictionary-based measures in performance which drew us to devise a corpus-based similarity measure that would be comparable to dictionary-based measures in performance.

Third, some existing corpus-based word segmentation methods provide good precision score, but provide low recall score and as a result low F-measure score. So, we were inspired to propose a corpus-based word segmentation method that would provide good F-measure score.

In this paper, we make the following contributions.

- First, we present a corpus-based method for determining the semantic similarity of two target words. Our method uses pointwise mutual information (PMI) to sort lists of important neighbor words of the two target words and distinguish the words which are common in both lists and aggregate their PMI values from the opposite list to calculate the relative similarity score. Evaluation results show that our method outperforms several competing corpus-based methods.
- Second, we present a new corpus-based method for automatic word segmentation. Our method uses corpus-type frequency information to choose the type with maximum length and frequency from "desegmented"[1] text. It also uses a modified forward–backward matching technique using maximum length frequency and entropy rate if any non-matching portions of the text exist. Evaluation results show that our method outperforms several competing corpus-based segmentation methods.
- Third, we present a name-based element-level schema matching method that exploits our proposed corpus-based word similarity and word segmentation method together with a substring matching algorithm. Finally, we point out some areas where these methods or modified versions of these methods can be exploited.

The remainder of this paper is organized as follows. Section 2 introduces the idea of corpus-based semantic similarity of words, and describes in detail our proposed method, with experimental results. In Sect. 3, we discuss corpus-based word segmentation and related work. We also describe our proposed word segmentation method with examples and evaluation. Then Sect. 4 presents database schema matching: a brief overview of schema matching approaches and our proposed name-based element-level hybrid schema matching method. Finally, we conclude in Sect. 5 with a brief discussion of future work.

2 Semantic similarity of words

Semantic relatedness refers to the degree to which two concepts or words are related (or not) whereas semantic similarity is a special case or a subset of semantic relatedness. Humans are able to easily judge if a pair of words are related in some way. For example, most would agree that *apple* and *orange* are more related than are *apple* and *toothbrush*. Budanitsky and Hirst [9] point out that semantic similarity is used when similar entities such as *apple* and *orange* or *table* and *furniture* are compared. These entities are close to each other in an *is–a* hierarchy. For example, *apple* and *orange* are hyponyms of *fruit* and *table* is a hyponym of *furniture*. However, even dissimilar entities may be semantically related, for example, *glass* and *water*, *tree* and *shade*, or *gym* and *weights*. In this case the two entities are intrinsically not

[1] Words are connected together to form a name.

similar, but are related by some relationship. Sometimes this relationship may be one of the classical relationships such as meronymy (is part of) as in *computer–keyboard* or a non-classical one as in *glass–water, tree–shade* and *gym–weights*. Thus two entities are semantically related if they are semantically similar (close together in the *is–a* hierarchy) or share any other classical or non-classical relationships.

Measures of the semantic similarity of words have been used for a long time in applications in natural language processing and related areas, such as the automatic creation of thesauri [23,35,37], automatic indexing, text annotation and summarization [36], text classification, word sense disambiguation [34,35,65], information extraction and retrieval [10, 62,64], lexical selection, automatic correction of word errors in text and discovering word senses directly from text [45].

A word similarity measure is also used for language modeling by grouping similar words into classes [8]. In databases, word similarity can be used to solve semantic heterogeneity, a key problem in any data sharing system whether it is a federated database, a data integration system, a message passing system, a web service or a peer-to-peer data management system [39].

2.1 Related work on semantic similarity of words

Many different measures of semantic similarity between word pairs have been proposed, some using statistical or distributional techniques [24,37], some using lexical databases (thesaurus) and some hybrid approaches, combining distributional and lexical techniques. PMI-IR [61] uses PMI and information retrieval (IR) to measure the similarity of pairs of words. PMI-IR is a statistical approach that uses a huge data source: the web and the PMI of two words are approximated by the number of web documents where they co-occur. Another well-known statistical approach to measuring semantic similarity is latent semantic analysis (LSA) [31]. We will briefly discuss these two approaches in next subsections.

Individual words in a given text corpus have more or less differing contexts around them. The context of a word is composed of words co-occurring with it within a certain window around it. Distributional measures use statistics acquired from a large text corpora to determine how similar the contexts of two words are. These measures are also used as approximations to measures of semantic similarity of words, because words found in similar contexts tend to be semantically similar. Such measures have traditionally been referred to as measures of distributional similarity. If two words have many co-occurring words, then similar things are being said about both of them and therefore they are likely to be semantically similar. Conversely, if two words are semantically similar then they are likely to be used in a similar fashion in text and thus end up with many common co-occurrences. For example, the semantically similar *car* and *vehicle* are expected to have a number of common co-occurring words such as *parking*, *garage*, *model, industry, accident*, *traffic* and so on, in a large enough text corpus.

Various distributional similarity measures were discussed in [63] where co-occurrence types of a target word are the contexts in which it occurs and these have associated frequencies which may be used to form probability estimates. Lesk [33] was one of the first to apply the cosine measure, which computes the cosine of the angle between two vectors, to word similarity. The Jensen–Shannon (JS) divergence measure [15,50] and the skew divergence measure [32] are based on the Kullback–Leibler (KL) divergence measure. Jaccard's coefficient [56] calculates the proportion of features belonging to either word that are shared by both words. In the simplest case, the features of a word are defined as the contexts in which it has been seen to occur. PMI was first used to measure word similarity by Church and Hanks [13] where positive values indicate that words occur together more than would be expected under an independence assumption and negative values indicate that one word tends to appear only when the other does not. Jaccard-MI is a variant [37] in which the features of a word are those contexts for which the pointwise mutual information between the word and the context is positive. Average mutual information corresponds to the expected value of two random variables using the same equation as PMI and was used as a word similarity measure by [15,52]. Cosine of pointwise mutual information was used by [45] to uncover word senses from text. L_1 norm method was proposed as an alternative word similarity measure in language modeling to overcome zero-frequency problems for bigrams [15]. A likelihood ratio was used by [20] to test word similarity under the assumption that the words in text have a binomial distribution.

There are several dictionary-based approaches to measuring the similarity of words. Most of the dictionary-based approaches use WordNet [43], a broad coverage lexical network of English words. Some use Roget's Thesaurus. Budanitsky and Hirst [9] presented a detail overview of several WordNet based measures. We briefly discuss Lin's [38] approach, one of the hybrid measures using both the WordNet and corpus in next subsection. Jarmasz and Szpakowicz [27] implemented a similarity measure using Roget's Thesaurus.

2.1.1 Latent semantic analysis (LSA)

LSA [31], a high-dimensional linear association model, analyzes a large corpus of natural text and generates 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 [31]. 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 corpus is broken up into chunks of texts approximately the size of a small text or paragraph. Landauer and Dumais mentioned in [31] "... we took a sample consisting of (usually) the whole text or its first 2,000 characters, whichever was less, for a mean text sample length of 151 words, roughly the size of a rather long paragraph''. Analyzing each text or paragraph, the number of occurrences of each word is set in a matrix with a column for each word and a row for each paragraph. Then each cell of the matrix (a word by context matrix, X), is transformed from the raw frequency count into the log of the count. After that each cell is divided by the entropy of the column, given by $-\sum p \log p$, where the summation is over all the paragraphs the word appeared.

The next step is to apply SVD to X, to decompose X into a product of three matrices

$$X = WSP',$$

where W and P are in column orthonormal form (i.e., columns are orthogonal) and S is the diagonal matrix of non-zero entries (singular values). To reduce dimensions, the rows of W and P corresponding to the highest entries of S are kept. In other words, the new lower dimensional matrices W_{L}, P_{L} and S_{L} are the matrices produced by removing the columns and rows with smallest singular values from W, P and S. This new matrix

$$X_{\mathrm{L}} = W_{\mathrm{L}} S_{\mathrm{L}} P'_{\mathrm{L}}$$

is a compressed matrix which represents all the words and text samples in a lower dimensional space. Then the similarity of two words, using LSA, is measured by the cosine of the angle between their corresponding row vectors.

2.1.2 PMI-IR

PMI-IR [61], a simple unsupervised learning algorithm for recognizing synonyms, uses PMI as follows:

$$\mathrm{score}(choice_i) = p(problem \ \& \ choice_i)/p(choice_i)$$

Here, *problem* represents the problem word and $\{choice_1, choice_2, \ldots, choice_n\}$ represent the alternatives. $p(problem \ \& \ choice_i)$ is the probability that *problem* and $choice_i$ co-occur. In other words, each choice is simply scored by the conditional probability of the problem word, given the choice word, $p(problem|choice_i)$. If *problem* and $choice_i$ are statistically independent, then the probability that they co-occur is given by the product $p(problem) \cdot p(choice_i)$. If they are not independent, and they have a tendency to co-occur, then $p(problem \ \& \ choice_i)$ will be greater than $p(problem) \cdot p(choice_i)$.

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:

$$\mathrm{score}_1(choice_i) = \mathrm{hits}(problem \ \mathrm{AND} \ choice_i)/\mathrm{hits}(choice_i)$$

Here, hits(x) is the number of hits (the number of documents retrieved) when the query x is given to AltaVista. AltaVista provides how many documents contain both *problem* and $choice_i$, and then how many documents contain $choice_i$ alone. The ratio of these two numbers is the score for $choice_i$. There are three other versions of this scoring equation based on the closeness of the pairs in documents, considering antonyms, and taking context into account.

2.1.3 Lin's measure

Lin [38] noticed that most of the similarity measures were tied to a particular application domain or resource and then he attempted to define a similarity measure that would be both universal and theoretically justified. He used the following three intuitions as a basis:

(1) The similarity between A and B is related to their commonality. The more commonality they share, the more similar they are. The commonality between A and B is measure by

$$I(\mathrm{common}(A, B))$$

where common(A, B) is a proposition that states the commonalities between A and B; $I(s)$ is the amount of information contained in a proposition s.

(2) The similarity between A and B is related to the differences between them. The more differences they have, the less similar they are. The difference between A and B is measure by

$$I(\mathrm{description}(A, B)) - I(\mathrm{common}(A, B))$$

where description(A, B) is a proposition that describes what A and B are.

(3) The maximum similarity between A and B is reached when A and B are identical, no matter how much commonality they share.

Given these assumptions and definitions and the apparatus of information theory, Lin proved the following theorem:

Similarity Theorem. *The similarity between A and B is measured by the ratio between the amount of information*

needed to state the commonality of A and B and the information needed to fully describe what A and B are:

$$sim(A, B) = \frac{\log P(common(A, B))}{\log P(description(A, B))}$$

Lin demonstrated how this similarity theorem could be applied in different domains using WordNet and corpus. For example, his measure of similarity between two concepts in taxonomy is a corollary of this theorem:

$$sim(A, B) = \frac{2 \log P(lso(A, B))}{\log P(A) + \log P(B)},$$

where probabilities $P(x)$ are determined by

$$P(x) = \frac{\sum_{w \in W(x)} count(w)}{N},$$

where $W(x)$ is the set of words (nouns) in the corpus whose senses are subsumed by concept x, and N is the total number of word (noun) tokens in the corpus that are also present in WordNet.

2.2 Proposed second-order co-occurrence PMI method

Let w_1 and w_2 be the two words for which we need to determine the semantic similarity and $C = \{c_1, c_2, \ldots, c_m\}$ denotes a large corpus of text (after some preprocessing, e.g., stop words elimination and lemmatization) containing m words (tokens). Also, let $T = \{t_1, t_2, \ldots, t_n\}$ be the set of all unique words (types) which occur in the corpus C. Unlike the corpus C, which is an ordered list containing many occurrences of the same words, T is a set containing no repeated words. Throughout this section, we will use w to denote either w_1 or w_2.

We set a parameter α, which determines how many words before and after the target word w, will be included in the context window. The window also contains the target word w itself, resulting in a window size of $2\alpha + 1$ words. The steps in determining the semantic similarity involve scanning the corpus and then extracting some functions related to frequency counts.

We define the *type frequency* function,

$$f^t(t_i) = |\{k\colon c_k = t_i\}|, \quad \text{where } i = 1, 2, \ldots, n$$

which tells us how many times the type t_i appeared in the entire corpus. Let

$$f^b(t_i, w) = |\{k\colon t_k = w \text{ and } t_{k \pm j} = t_i\}|,$$

where $i = 1, 2, \ldots, n$ and $-\alpha \leq j \leq \alpha$, be the *bigram frequency* function. $f^b(t_i, w)$ tells us how many times word t_i appeared with word w in a window of size $2\alpha + 1$ words.

Then we define *pointwise mutual information* function for only those words having $f^b(t_i, w) > 0$,

$$f^{\text{pmi}}(t_i, w) = \log_2 \frac{f^b(t_i, w) \times m}{f^t(t_i) f^t(w)},$$

where $f^t(t_i) f^t(w) > 0$ and m is total number of tokens in corpus C as mentioned earlier. Now, for word w_1, we define a set of words, X, sorted in descending order by their PMI values with w_1 and take the top-most β_1 words having $f^{\text{pmi}}(t_i, w_1) > 0$.

$$X = \{X_i\}, \quad \text{where } i = 1, 2, \ldots, \beta_1 \\ \text{and } f^{\text{pmi}}(t_1, w_1) \geq f^{\text{pmi}}(t_2, w_1) \geq \cdots f^{\text{pmi}}(t_{\beta 1 - 1}, w_1) \\ \geq f^{\text{pmi}}(t_{\beta 1}, w_1)$$

Similarly, for word w_2, we define a set of words, Y, sorted in descending order by their PMI values with w_2 and take the top-most β_2 words with $f^{\text{pmi}}(t_i, w_2) > 0$.

$$Y = \{Y_i\}, \quad \text{where } i = 1, 2, \ldots, \beta_2 \\ \text{and } f^{\text{pmi}}(t_1, w_2) \geq f^{\text{pmi}}(t_2, w_2) \geq \cdots f^{\text{pmi}}(t_{\beta 2 - 1}, w_2) \\ \geq f^{\text{pmi}}(t_{\beta 2}, w_2)$$

Note that we have not yet determined the value for βs (either β_1 or β_2) which actually depend on the word w and the number of types in the corpus (this will be discussed in the next section).

Again, we define the *β-PMI summation* function. For word w_1, the *β-PMI summation* function is:

$$f^{\beta}(w_1) = \sum_{i=1}^{\beta_1} \left(f^{\text{pmi}}(X_i, w_2) \right)^{\gamma},$$

where $f^{\text{pmi}}(X_i, w_2) > 0$ and $f^{\text{pmi}}(X_i, w_1) > 0$ which sums all the positive PMI values of words in the set Y also common to the words in the set X. In other words, this function actually aggregates the positive PMI values of all the semantically close words of w_2 which are also common in w_1. Note that we call it semantically close because all these words have high PMI values with w_2 and this does not ensure the closeness with respect to the distance within the window size.

Similarly, for word w_2, the *β-PMI summation* function is:

$$f^{\beta}(w_2) = \sum_{i=1}^{\beta_2} \left(f^{\text{pmi}}(Y_i, w_1) \right)^{\gamma},$$

where $f^{\text{pmi}}(Y_i, w_1) > 0$ and $f^{\text{pmi}}(Y_i, w_2) > 0$ which sums all the positive PMI values of words in the set X also common to the words in the set Y . In other words, this function aggregates the positive PMI values of all the semantically close words of w_1 which are also common in w_2. We have not discussed the criteria for choosing the exponential parameter γ (this will be discussed in the next subsection).

Finally, we define the *semantic PMI similarity* function between two words, w_1 and w_2,

$$Sim(w_1, w_2) = \frac{f^{\beta}(w_1)}{\beta_1} + \frac{f^{\beta}(w_2)}{\beta_2}$$

2.2.1 Choosing the values of β and γ

A rule of thumb is used to choose the value of β. The value of β is related to how many times the word w appears in the corpus; that is, the frequency of w as well as the number of types in the corpus. We define β as

$$\beta_i = \left(\log(f^t(w_i))\right)^2 \frac{(\log_2(n))}{\delta},$$

where $i = 1, 2$ and δ is a constant.

For all of our experiments we used $\delta = 6.5$. The value of δ depends on the size of the corpus. The smaller the corpus we use, the smaller the value of δ we should choose. If we lower the value of β we lose some important/interesting words, and if we increase it we consider more words common to both w_1 and w_2 and this significantly degrades the result.

γ should have a value greater than 1. The higher we choose the value of γ, the greater emphasis on words having very high PMI values with w. For all our experiments, we chose $\gamma = 3$. We experimented on a small portion of the BNC to find out the value of δ and γ. The value $\gamma \geq 4$ is not a good choice because it puts too much emphasis on words that have very high PMI values with w and ignores all the words having moderate or low PMI values.

2.3 Experimental results

Our method was empirically evaluated on the task of solving 80 synonym TOEFL questions and 50 synonym ESL questions; and using Miller and Charles' [42] 30 noun pairs subset and Rubenstein and Goodenough's [53] 65 noun pairs.

In literature, though most of the word similarity measures were evaluated using Miller and Charles' [42] 30 noun pairs subset and Rubenstein and Goodenough's [53] 65 noun pairs, we also evaluated our method on the 80 synonym TOEFL questions and 50 synonym ESL questions to judge how well our method performs on a task-based test.

We computed the SOC-PMI similarity values using the British National Corpus (BNC)[2] as a source of frequencies and contexts. The size of this corpus is approximately 100 million words, and it is a balanced corpus: it contains texts from various sources, general British English.

Landauer and Dumais [31] employed word similarity measures to answer 80 synonym test questions from the Test of English as a Foreign Language (TOEFL) using Latent Semantic Analysis (LSA). Turney [61] applied his PMI-IR measure to answer 50 synonym test questions from a collection of English as a Second Language (ESL) tests and the same 80 TOEFL questions set that Landauer and Dumais [31] used.

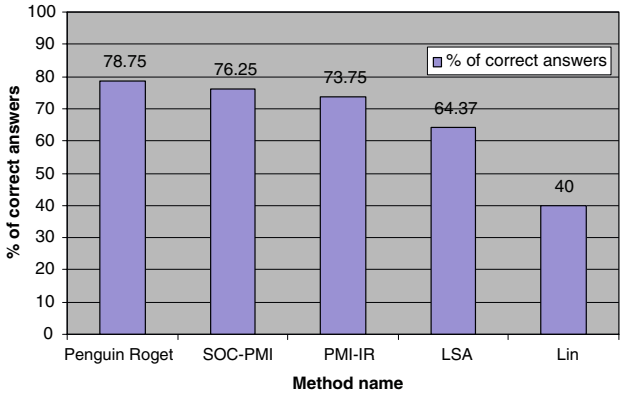

Fig. 1 Results on the 80 TOEFL questions

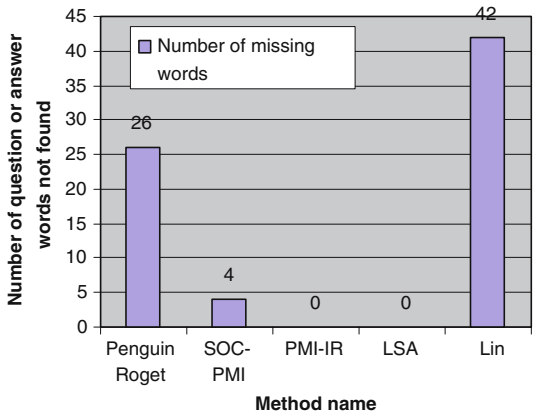

Fig. 2 Number of missing words on the 80 TOEFL questions

For the 80 TOEFL questions, the SOC-PMI method correctly answered 76.25% of the questions, as shown in Fig. 1. This is an improvement over the results presented by Landauer and Dumais [31], using LSA, where 64.37% of the questions were answered correctly, and Turney [61], using the PMI-IR algorithm, where the best result was 73.75%. A human average score on the same question set is 64.5% [31].

For the 50 ESL questions, the SOC-PMI method correctly answered 68% of the questions (without using the context) compared to [61] where the best result was 66%, as shown in Fig. 3. The number of question or answer words that different methods did not find for the 80 TOEFL questions and 50 ESL questions are in Figs. 2 and 4, respectively. These questions were not answered in the experiments because the similarity could not be computed for all pairs of words.

For Miller and Charles' [42] dataset, we got a correlation of 0.764 with the human judges (Fig. 5). For Rubenstein and Goodenough's [53] dataset we got a correlation of 0.729.

[2] http://www.natcorp.ox.ac.uk/.

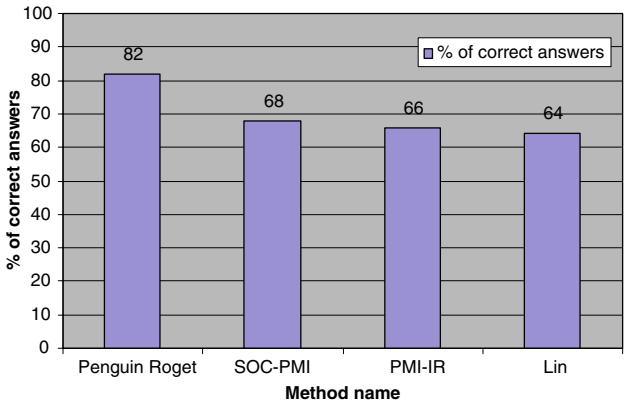

Fig. 3 Results on the 50 ESL questions

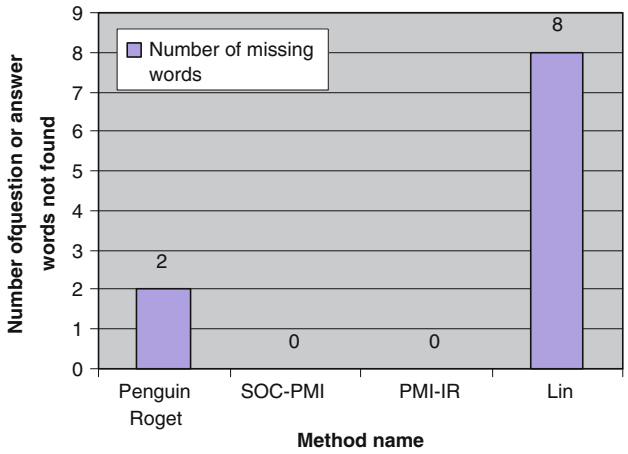

Fig. 4 Number of missing words on the 50 ESL questions

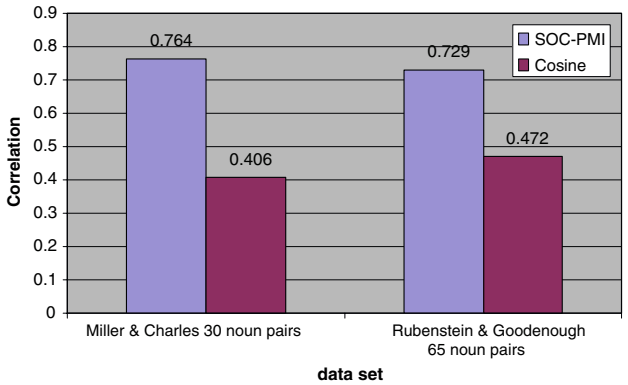

Fig. 5 Correlation of noun pairs

These correlation values are very good for a corpus-based measure, considering that a baseline vector space method using cosine obtains 0.406 for the first set and 0.472 for the second set. For dictionary-based measures [27], the correlations are slightly higher, but comparable to ours.

Figures 1 and 3 show that a method using Roget's thesaurus provides 2.5 and 14% more correct results than ours for the 80 TOEFL questions and 50 ESL questions, respectively. The WordNet-based measures—implemented in the WordNet::Similarity package by Pedersen et al. [46]—achieve lower accuracy on the two data sets than the Roget measure [27]. The fact that the Roget measure performs better than the corpus-based measures is to be expected, because Roget's thesaurus can be seen as a classification system. It is composed of six primary classes and each is composed of multiple divisions and then sections. This may be conceptualized as a tree containing over a 1,000 branches for individual *meaning clusters* or semantically linked words. These words are not exactly synonyms, but can be viewed as colors or connotations of a meaning or as a spectrum of a concept. One of the most general words is chosen to typify the spectrum as its headword, which labels the whole group.

Second-order co-occurrence PMI may be helpful as a tool to aid in the automatic construction of the synonyms of a word. This could be a future application of our proposed method.

PMI-IR used AltaVista's Advanced Search query syntax to calculate probabilities. Note that the "NEAR" search operator of AltaVista is an essential operator in the PMI-IR method. However, it is no longer in use in AltaVista; this means that, from the implementation point of view, it is not possible to use the PMI-IR method in the same form in new systems. In any case, from the algorithmic point of view, the advantage of using SOC-PMI in our system is that it can calculate the similarity between two words that do not co-occur frequently, because they co-occur with the same neighboring words.

Detecting semantic outliers in speech recognition transcripts can use semantic similarity measures [26] and a corpus-based similarity measure plays an important role because of its large type coverage. The corpus-based measures were shown to perform better than the Roget-based measure in the task. In future work we can try the SOC-PMI method for this task.

3 Corpus-based word segmentation

Word segmentation is an important problem in many natural language processing tasks; for example, in speech recognition where there is no explicit word boundary information given within a continuous speech utterance, or in interpreting written languages such as Chinese, Japanese and Thai where words are not delimited by white-space but instead must be inferred from the basic character sequence. We differentiate the terms *word breaking* and *word segmentation*. Word breaking refers to the process of segmenting known words that are predefined in a lexicon. Word segmentation

refers to the process of both lexicon word segmentation and unknown word or new word[3] detection. Automatic word segmentation is a basic requirement for unsupervised learning in morphological analysis. Developing a morphological analyzer for a new language by hand can be costly and time consuming, requiring a great deal of effort by highly specialized experts.

In databases, word segmentation can be used in schema matching to solve semantic heterogeneity, a key problem in any data sharing system whether it is a federated database, a data integration system, a message passing system, a web service or a peer-to-peer data management system [39]. The name of an element in a database typically contains words that are descriptive of the element's semantics. N-grams[4] have been shown to work well in the presence of short forms, incomplete names and spelling errors that are common in schema names [19].

Also, extracting words (word segmentation) from a scanned document page or a PDF is an important and basic step in document structure analysis and understanding systems; incorrect word segmentation during OCR leads to errors in information retrieval and in understanding the document.

One of the common approaches involving an extensive word list combined with an informed segmentation algorithm can help achieve a certain degree of accuracy in word segmentation, but the greatest barrier to accurate word segmentation is in recognizing unknown words, words not in the lexicon of the segmenter. This problem is dependent both on the source of the lexicon as well as the correspondence between the text in question and the lexicon. Fung and Wu [21] reported that segmentation accuracy is significantly higher when the lexicon is constructed using the same type of corpus as the corpus on which it is tested.

The term *maximum-length descending-frequency* means that we choose maximum length n-grams that have a minimum threshold frequency and then we look for further n-grams in descending order based on length. If two n-grams have same length then we choose the n-gram with higher frequency first and then the n-gram with next higher frequency if any of its characters are not a part of the previous one. If we follow this procedure, after some iterations, we can be in a state with some remaining characters (we call it *residue*) that are not matched with any type in the corpus. To solve this, we use the *leftMaxMatching* and *rightMaxMatching* algorithms presented in Sect. 3.2 along with entropy rate.

[3] New words in this paper refer to out-of-vocabulary words that are neither recognized as named entities or factoids, nor derived by morphological rules. These words are mostly domain-specific and/or time-sensitive.

[4] Sequence of n consecutive characters.

3.1 Related work on word segmentation

Word segmentation methods can be roughly classified as either dictionary-based or statistically 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 also contain special components for unknown word identification. In particular, statistical methods have been widely applied because they use a probabilistic or cost-based scoring mechanism rather than a dictionary to segment the text [22].

A simple word segmentation algorithm is to consider each character a distinct word. This is practical for Chinese because the average word length is very short, usually between one and two characters, depending on the corpus [21], and actual words can be recognized with this algorithm. Although it does not assist in task such as parsing, part-of-speech tagging or text-to-speech systems [60], the character-as-word segmentation algorithm has been used to obtain good performance in Chinese information retrieval, a task in which the words in a text play a major role in indexing.

One of the most popular methods is *maximum matching* (MM), usually augmented with heuristics to deal with ambiguities in segmentation. Another very common approach to word segmentation is to use a variation of the *maximum matching algorithm*, frequently referred to as the *greedy algorithm*. The greedy algorithm starts at the first character in a text and, using a word list for the language being segmented, attempts to find the longest word in the list starting with that character. If a word is found, the maximum-matching algorithm marks a boundary at the end of the longest word, then begins the same longest match search starting at the character following the match. If no match is found in the word list, the greedy algorithm simply segments that character as a word and begins the search starting at the next character. A variation of the greedy algorithm segments a sequence of unmatched characters as a single word; this variant is more likely to be successful in writing systems with longer average word lengths. In this manner, an initial segmentation can be obtained that is more informed than a simple character-as-word approach. As a demonstration of the application of the character-as-word and greedy algorithms, consider an example of "desegmented" English, in which all the white space has been removed: the "desegmented" version of the text, *the most favourite music of all time*, would thus be *themostfavouritemusicofalltime*, Applying the character-as-word algorithm would result in the useless sequence of tokens *t h e m o s t f a v o u r i t e m u s i c o f a l l t i m e*, which is why this

algorithm only makes sense for languages such as Chinese. Applying the greedy algorithm with a "perfect" word list containing all known English words would first identify the word *them*, since that is the longest sequence of letters starting at the initial *t* which forms an actual word. Starting at the *o* following *them*, the algorithm would then find no match. Continuing in this manner, *themostfavouritemusicofalltime* would be segmented by the greedy algorithm as *them o s t favourite music of all time*. A variant of the maximum matching algorithm is the *reverse maximum matching* algorithm, in which the matching proceeds from the end of the string of characters, rather than the beginning. In the foregoing example, *themostfavouritemusicofalltime* would be segmented as *the most favourite music o fall time* by the reverse maximum matching algorithm. Greedy matching from the beginning and the end of the string of characters enables an algorithm such as *forward–backward matching*, in which the results are composed and the segmentation optimized based on the two results [16].

Many unsupervised methods have been proposed for segmenting raw character sequences with no boundary information into words [4,5,11,12,17,25,29]. Brent [4] gives a good survey of these methods. Most current approaches are using some form of expectation maximization (EM) to learn a probabilistic speech-or-text model and then employing Viterbi decoding procedures [48] to segment new speech or text into words. One reason that EM is widely adopted for unsupervised learning is that it is guaranteed to converge to a good probability model that locally maximizes the likelihood or posterior probability of the training data. For the problem of word segmentation, EM is typically applied by first extracting a set of candidate multi-grams from a given training corpus [17], initializing a probability distribution over this set, and then using the standard iteration to adjust the probabilities of the multi-grams to increase the posterior probability of the training data.

Saffran et al. [55] proposed that word segmentation from continuous speech may be achieved using transitional probabilities (TP) between adjacent syllables A and B, where, $\mathrm{TP}(A \rightarrow B) = \mathrm{P}(AB)/\mathrm{P}(A)$, with $\mathrm{P}(AB)$ being the frequency of B following A, and $\mathrm{P}(A)$ the total frequency of A. Word boundaries are postulated at local minima, where the TP is lower than its neighbors.

In corpus-based word segmentation, there is either no explicit model learnt, as when neural networks [54] or lazy learning [14] are used, or the derived models are less sophisticated and do not use any abstractions of the word constituents found in data [7,41]. Using annotated corpora greatly facilitates learning. However, there are situations in which one is interested in unsupervised learning (UL), that is, from unannotated corpora. Motivation for UL can vary from purely pragmatic, such as the high cost or unavailability of annotated corpora, to theoretical, when language is modeled as yet another communication code within the framework of information theory [58].

3.2 Proposed word segmentation method

Let $S = l_1 l_2 l_3 \ldots l_m$ denote a text of m consecutive characters without any space in between them for which we need to segment and $C = \{c_1, c_2, \ldots, c_\tau\}$ denote a large corpus of text containing τ words (tokens). Also, let $T^p = \{t_1, t_2, \ldots, t_p\}$ be the set of all (p) unique words (types) which occur in the corpus C and $T^f = \{f_1, f_2, \ldots, f_p\}$ be the set of frequencies of all the corresponding types in T^p; where f_x is the frequency of type t_x. Unlike the corpus C, which is an ordered list containing many occurrences of the same words, T^p is a set containing no repeated words. Again, let n be the maximum length of any possible words in the segmented words list where $n \leq m$ and $N^p = \{l_1, l_2, \ldots, l_n, l_1 l_2, l_2 l_3, \ldots, l_1 l_2 \ldots l_n, \ldots\}$ be the set of all possible n-grams where $\eta = |N^p|$ is the total number of n-grams in N^p. We can also consider N^p as $N^p = \{w_1, w_2 \ldots, w_\eta\}$. And $N^f = \{f_1', f_2' \ldots, f_\eta'\}$ be the set of frequencies of all the corresponding n-grams of N^p taken from T^f; where f_x' is the frequency of w_x. To get rid of the noise types of the corpus, we assign a set of minimum frequencies for each possible length from 1 to n to be considered as a valid word. $M^f = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, where α_x is the minimum frequency required to be a valid word[5] of length x. Minimum required frequency, α_x is inversely proportional to the word length, x. The steps of the method are as follows:

Step 1: Sort all the elements of N^p in descending order based on length (in characters). Again sort in descending order for same length words of the sorted N^p (say $\overline{N^p}$) based on the frequencies of N^f. For each element in $\overline{N^p}$ do the next steps:

Step 2: If $S \neq \emptyset$ and the current maximum length n-gram (say w_n) in $\overline{N^p}$ satisfies $f_n^{'} \geq \alpha_{|w_n|}$ and $w_n \in S$ (i.e., $S \cap w_n = w_n$) then add w_n to segmented word list, $S^{'}$ (i.e., $S' \leftarrow S^{'} \cup w_n$) and remove w_n from S (i.e., $S \leftarrow S \backslash w_n$) and add a blank space as a boundary mark.

Step 3: If $S \neq \emptyset$ and not all elements in $\overline{N^p}$ are done then update w_n by the next maximum length n-gram from $\overline{N^p}$ and go to step 2.

Step 4: Rearrange all the words of $S^{'}$ in accordance with S. If $S = \emptyset$, then output $S^{'}$ and exit. Otherwise, for each remaining chunks[6], r in S call *matchResidue*(r), output $S^{'}$ and exit.

[5] We experiment in BNC and find out the minimum frequency range for nearly all valid words up to words of length 34, as in BNC the length of the longest valid word is 34. For $n = 34$, $M^f = \{1000, 500, 50, 16, 15, 12, 10, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2\}$; for example, nearly all valid words of length 3 in BNC have a minimum frequency of 50 or more.

[6] A single chunk may contain one or more characters.

Fig. 6 Matching residue

Algorithm ***matchResidue***

Input: r, $S^{'}$ // r is the remaining chunk and $S^{'}$ is the current segmented word list

1. // Take the prefix word, w_{n-1} and suffix word, w_n of r from $S^{'}$
2. // according to the would be position of r in $S^{'}$.
3. $S^{'} \leftarrow S^{'} \setminus w_{n-1}$
4. $S^{'} \leftarrow S^{'} \setminus w_n$
5. $S_t \leftarrow w_{n-1} \cup r \cup w_n$
6. // $S_t = \{l_1 l_2 l_3 \ldots l_m\}$, where m is the length of S_t
7. $S_t^{'} \leftarrow leftMaxMatching(S_t)$
8. $S_t^{''} \leftarrow rightMaxMatching(S_t)$
9. **if** ($|S_t^{'}| > |S_t^{''}|$)
10. $S^{'} \leftarrow S^{'} \cup S_t^{''}$
11. **elseif** ($|S_t^{'}| < |S_t^{''}|$)
12. $S^{'} \leftarrow S^{'} \cup S_t^{'}$
13. **else**
14. find a $x \in \{S_t^{'}, S_t^{''}\}$ for which entropy rate $\frac{1}{|x|}\sum_{i=1}^{|x|}\log_2(f_i)$ is maximum
15. $S^{'} \leftarrow S^{'} \cup x$
16. **end**

Output: $S^{'}$ // $S^{'}$ is the segmented word list after matching residue

In ***matchResidue***, if ***leftMaxMatching*** and ***rightMaxMatching*** return same numbers of words then we use entropy rate to decide which set of words we will accept. The intuition behind using entropy rate is that if we have a set of words having larger average frequency (we use 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 (Figs. 6, 7, 8).

3.3 A walk-through example

As a demonstration of the application of the proposed algorithms, consider the same example of "desegmented" English text, $S = \{themostfavouritemusicofalltime\}$[7]. We have used the BNC corpus to calculate T^p and T^f. Let $n = 9$ be the maximum length[8] of all possible word in S and $M^f = \{1000, 500, 50, 16, 15, 12, 10, 3, 2\}$. Table 1 shows the sorted n-grams, $\overline{N^p}$ and their frequencies, N^f for this specific example.

For each element w_n (say, *favourite*) in $\overline{N^p}$,

Step 2: w_n satisfies $f_n^{'} \geq \alpha_{|w_n|}$ as $4671 \geq 2$ and w_n is a substring of S.
$S^{'} = \{favourite\}$ and $S = \{themost musicofalltime\}$.
Step 3: Not all elements in $\overline{N^p}$ are done, update $w_n = \{alltime\}$ and go to step 2.
Step 2: does not satify $f_n^{'} \geq \alpha_{|w_n|}$ as $6 < 10$ though w_n is a substring of S.
Step 3: Not all elements in $\overline{N^p}$ are done, update $w_n = \{favour\}$ and go to step 2.
Step 2: Condition fails as w_n is not a substring of S.
Step 3: Not all elements in $\overline{N^p}$ are done, update $w_n = \{musico\}$ and go to step 2.
Step 2: Condition fails as w_n does not satisfy $f_n^{'} \geq \alpha_{|w_n|}$ as $10 < 12$.
Step 3: Not all elements in $\overline{N^p}$ are done, update $w_n = \{music\}$ and go to step 2.
Step 2: w_n satisfies $f_n^{'} \geq \alpha_{|w_n|}$ as $15134 \geq 15$ and w_n is a substring of S.
$S^{'} = \{favourite, music\}$ and
$S = \{themost ofalltime\}$.
We will only show the step 2 of all the remaining elements in $\overline{N^p}$ that satisfy the conditions.
Step 2: $w_n = \{them\}$
$S^{'} = \{favourite, music, them\}$ and $S = \{ost ofalltime\}$.
Step 2: $w_n = \{time\}$

[7] S is a set with one string element; a space in the string will be used to replace a substring that will be taken out, in order to distinguish the next parts to be processed.

[8] Though in BNC, the length of the longest valid word is 34.

Fig. 7 Matching leftmax

Algorithm *leftMaxMatching*

// n is the maximum length of any possible valid words in S_t and $n \leq m$

Input: S_t // S_t is a ìdese gmented" word

1. **while** $S_t \neq \emptyset$ do
2. $N^p \leftarrow \{l_1, l_1l_2, l_1l_2l_3 .., l_1l_2 \ldots l_n\}$;
3. that is, $N^p \leftarrow \{w_1, w_2 \ldots, w_n\}$
4. $N^f \leftarrow \{f_1^{'}, f_2^{'} \ldots, f_n^{'}\}$
5. $M^f \leftarrow \{\alpha_1, \alpha_2 \ldots, \alpha_n\}$
6. $i \leftarrow 1$
7. **while** ($i \leq n$ && $i \leq m$)
8. **if** ($f_i^{'} \geq \alpha_i$)
9. $max \leftarrow i$
10. **end**
11. increment i
12. **end**
13. $S_t^{'} \leftarrow S_t^{'} \cup w_{max}$
14. $S_t \leftarrow S_t \setminus w_{max}$
15. **end**

Output: $S_t^{'}$ // $S_t^{'}$ is the segmented word list after leftmax matching

Fig. 8 Matching rightmax

Algorithm *rightMaxMatching*

// n is the maximum length of any possible valid words in S_t and $n \leq m$

Input: S_t // S_t is a "desegmented" word

1. **while** $S_t \neq \emptyset$ do
2. $N^p \leftarrow \{l_m, l_{m-1}l_m, l_{m-2}l_{m-1}l_m, \ldots, l_{m-n}l_{m-n+1} \ldots l_m\}$;
3. that is, $N^p \leftarrow \{w_1, w_2 \ldots, w_n\}$
4. $N^f \leftarrow \{f_1^{'}, f_2^{'} \ldots, f_n^{'}\}$
5. $M^f \leftarrow \{\alpha_1, \alpha_2 \ldots, \alpha_n\}$
6. $i \leftarrow 1$
7. **while** ($i \leq n$ && $i \leq m$)
8. **if** ($f_i^{'} \geq \alpha_i$)
9. $max \leftarrow i$
10. **end**
11. increment i
12. **end**
13. $S_t^{'} \leftarrow S_t^{'} \cup w_{max}$
14. $S_t \leftarrow S_t \setminus w_{max}$
15. **end**

Output: $S_t^{'}$ // $S_t^{'}$ is the segmented word list after rightmax matching

$S^{'} = \{favourite, music, them, time\}$ and $S = \{ost\ of\ all\}$.

Step 2: $w_n = \{fall\}$

$S^{'} = \{favourite, music, them, time, fall\}$ and $S = \{ost\ o\}$.

Step 4: Rearrange $S^{'} = \{them, favourite, music, fall, time\}$ and $S \neq \emptyset$, so call ***matchResidue***(*ost*) and then ***matchResidue***(*o*).

Table 1 Sorted n-grams and their frequencies

N^{ps}	N^{f}	N^{ps}	N^{f}
favourite	4671	tem	31
alltime	6	emo	20
favour	6805	ost	18
musico	10	of	3052752
Music	15134	it	1054552
them	167457	he	641236
time	164294	me	131869
most	98276	us	80206
fall	11202	co	17476
item	3780	th	16486
rite	293	st	15565
allt	28	al	7299
emus	14	fa	2172
musi	3	em	1641
the	6057315	os	1005
all	282012	te	831
our	93463	si	658
tim	3401	mo	639
hem	305	ti	615
sic	292	im	576
mus	269	lt	485
emu	247	av	291
ico	95	mu	276
uri	46	ll	233
fal	44	ri	230
ofa	36	ou	151
mos	36	a	2179299
fav	33	i	873059

Case 1: ***matchResidue***(ost) is called

$$
\begin{aligned}
S^{'} &= S^{'}\backslash\{w_{n-1}, w_n\}\\
S^{'} &= \{them, favourite, music, fall, time\}\\
&\quad \backslash\{them, favourite\}\\
&= \{music, fall, time\}\\
S_t &= \{themostfavourite\}\\
S_t^{'} &= \{them, os, t, favourite\}\\
&\quad \leftarrow \textbf{\textit{leftMaxMatching}}(themostfavourite)\\
S_t^{''} &= \{the, most, favourite\}\\
&\quad \leftarrow \textbf{\textit{rightMaxMatching}}(themostfavourite)
\end{aligned}
$$

As $|S_t^{'}| > |S_t^{''}|$, $S^{'} = \{music, fall, time\} \cup S_t^{''}$;

$$S^{'} = \{THE, most, favourite, music, fall, time\}$$

Case 2: ***matchResidue***(o) is called

$$
\begin{aligned}
S^{'} &= S^{'}\backslash\{w_{n-1}, w_n\}\\
S^{'} &= \{the, most, favourite, music, fall, time\}\\
&\quad \backslash\{music, fall\}\\
&= \{the, most, favourite, time\}\\
S_t &= \{musicofall\}\\
S_t^{'} &= \{music, of, all\}\\
&\quad \leftarrow \textbf{\textit{leftMaxMatching}}(musicofall)\\
S_t^{''} &= \{mus, ico, fall\}\\
&\quad \leftarrow \textbf{\textit{rightMaxMatching}}(musicofall)
\end{aligned}
$$

As in this case $|S_t^{'}| = |S_t^{''}|$, we need to find whether $S_t^{'}$ or $S_t^{''}$ maximizes the entropy rate, $\frac{1}{|x|}\sum_{i=1}^{|x|}\log_2(f_i)$, where $x \in \{S_t^{'}, S_t^{''}\}$. The entropy rate for $S_t^{'}$ is (13.89 + 21.54 + 18.11)/3 and for $S_t^{'}$ is (8.07 + 6.57 + 13.45)/3. So, $S^{'} = \{the, most, favourite, time\} \cup S_t^{'}$, as $\frac{1}{\left|S_t^{'}\right|}\sum_{i=1}^{\left|S_t^{'}\right|}\log_2(f_i) > \frac{1}{\left|S_t^{''}\right|}\sum_{i=1}^{\left|S_t^{''}\right|}\log_2(f_i)$. Finally, $S^{'} = \{the, most, favourite, music, of, all, time\}$.

3.4 Evaluation and experimental results

An obstacle to high-accuracy word segmentation is that there are no widely accepted guidelines for what constitutes a word; therefore, there is no agreement on how to "correctly" segment a text in a "desegmented" language. Native speakers of a language do not always agree about the "correct" segmentation, and the same text could be segmented into several very different (and equally correct) sets of words by different native speakers. Such ambiguity in the definition of what constitutes a word makes it difficult to evaluate segmentation algorithms that follow different conventions, as it is nearly impossible to construct a "gold standard" against which to directly compare results [16]. As shown in Sproat et al. [59], the rate of agreement between two human judges on this task is less than 80%.

The performance of word segmentation is usually measured using *precision* and *recall*, where recall is defined as the percentage of words in the manually segmented text identified by the segmentation algorithm and precision is defined as the percentage of words returned by the algorithm that also occurred in the hand-segmented text in the same position. In general, it is easy to obtain high performance for one of the two measures but relatively difficult to obtain high performance for both. *F-measure* (F) is the geometric mean of *precision* (P) and *recall* (R) and expresses a trade-off between those two measures. These performance measures are defined

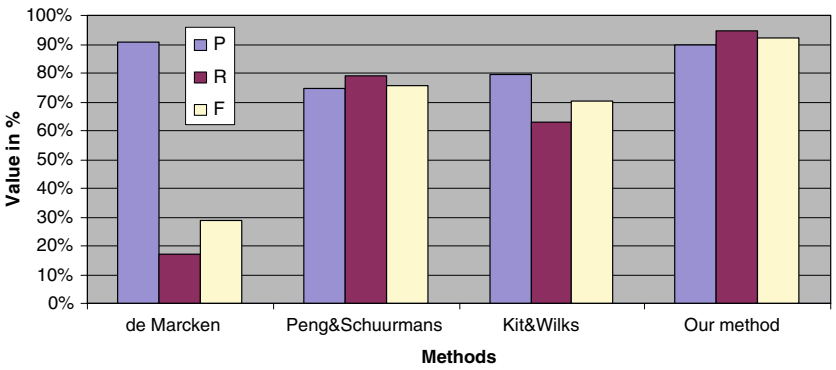

Fig. 9 Test result on the Brown corpus

as follows:

$$P = TP/(TP + FP)$$
$$R = TP/(TP + FN)$$
$$F = (1 + \beta)PR/(\beta P + R)$$

$= 2\ PR\ /\ (P + R)$, with $\beta = 1$ such that *precision* and *recall* weighted equally. Here, *TP*, *FP* and *FN* stand for True Positive, False Positive and False Negative respectively.

For instance, if the target segmentation is "we are human", and the model outputs "weare human", then precision is 1/2 ("human" out of "weare" and "human", recall is 1/3 ("human" out of "we", "are", and "human") and F-measure is 2/5.

We used the type frequency from BNC and tested our segmentation method on part of the BNC corpus. Specifically, we converted a portion of the corpus to lowercase letters and removed all white space and punctuation. We used 285K characters and 57,904 tokens as our test data. We obtained 84.28% word precision rate 81.63% word recall rate and 82.93% word F-measure.

In a second test, we used the type frequency from BNC and tested our segmentation method on the Brown corpus to make sure that we test on different vocabulary from the training data. This ensures that some of the word in the test set were not previously seen (out-of-vocabulary words). There were 4,705,022 characters and 1,003,881 tokens in the Brown corpus. We obtained 89.92% word precision rate, 94.69% word recall rate and 92.24% word F-measure. The average number of tokens per line could be the reason for obtaining better result when we tested on the Brown corpus, as 8.49 and 16.07 are the average number of tokens per line in the Brown corpus and the BNC corpus, respectively.

One of the best known results on segmenting the Brown corpus is due to Kit and Wilks [29] who use a description-length gain method. They trained their model on the whole corpus (6.13M) and reported results on the *training* set, obtaining a boundary precision of 79.33%, a boundary recall of 63.01% and boundary F-measure of 70.23%. Peng and Schuurmans [47] trained their model on a subset of the corpus (4,292K) and tested on *unseen* data. After the lexicon is optimized, they obtained 16.19% higher recall and 4.73% lower precision; resulting in an improvement of 5.2% in boundary F-measure. de Marcken [18] also used a minimum description length (MDL) framework and a hierarchical model to learn a word lexicon from raw speech. However, this work does not explicitly yield word boundaries, but instead recursively decomposes an input string down to the level of individual characters. As pointed out by Brent [4], this study gives credit for detecting a word if any node in the hierarchical decomposition spans the word. Under this measure, de Marcken [18] reports a word recall rate of 90.5% on the Brown corpus. However, his method creates numerous chunks and therefore only achieves a word precision rate of 17%. Christiansen et al. [12] used a simple recurrent neural network approach and report a word precision rate of 42.7% and word recall rate of 44.9% on spontaneous child-directed British English. Brent and Cartwright [5] used a MDL approach and reported a word precision rate of 41.3% and a word recall rate of 47.3% on the CHILDES collection. Brent [4] achieved about 70% word precision and 70% word recall by employing additional language modeling and smoothing techniques. Peng and Schuurmans [47] obtained 74.6% word precision rate, 79.2% word recall rate and 75.4% word F-measure on the Brown corpus. A balance of high precision and high recall is the main advantage of our proposed method. However, it is difficult to draw a direct comparison between these results because of the different test corpora used by different authors. Figure 9 summarizes the result of different methods which are tested on the Brown corpus based on precision, recall and F-measure. Though all the methods in Fig. 9 use the Brown corpus, the testing data sets in the Brown corpus are not exactly the same.

Actually, this method can effectively distill new words, special terms and proper nouns when the corpus covers a huge collection of both domain-dependent and domain-independent words, and it can effectively avoid statistical errors in shorter strings which belong to a longer one. However, names are not always easy to exploit and contain abbreviations and special characters that vary between domains.

This method can be used to address this issue, an important step of schema matching in databases. A generalized characteristic of this method is that it can be extended as a dictionary-based method or hybrid method with some additions to the algorithms. The absence of type frequencies in dictionary means that we can only use the length of the types. In that case, we need to focus on what type to choose for the same length types that share some common characters. Again, we cannot choose whether we take the elements of *leftMaxMatching* or *rightMaxMatching* when both of them return the same number of elements as we cannot use entropy rate for the absence of type frequencies. Experimental results show that our method can segment words with high precision and high recall.

4 Schema matching

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 can 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 [28]. Rahm and Bernstein [49] point out that it is not possible to determine fully automatically all 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.

4.1 Classification of schema matching approaches

Rahm and Bernstein [49] classify the major approaches to schema matching. Figure 10 shows part of their classification together with some sample approaches. An implementation of matching may use multiple match algorithms or *matchers*.

This allows selecting the matchers depending on the application domain and schema types. Given that multiple matchers could be used, two sub-problems arise. First, there is the realization of individual matchers, each of which computes a mapping based on a single matching criterion. Second, there is the combination of individual matchers, either 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*.

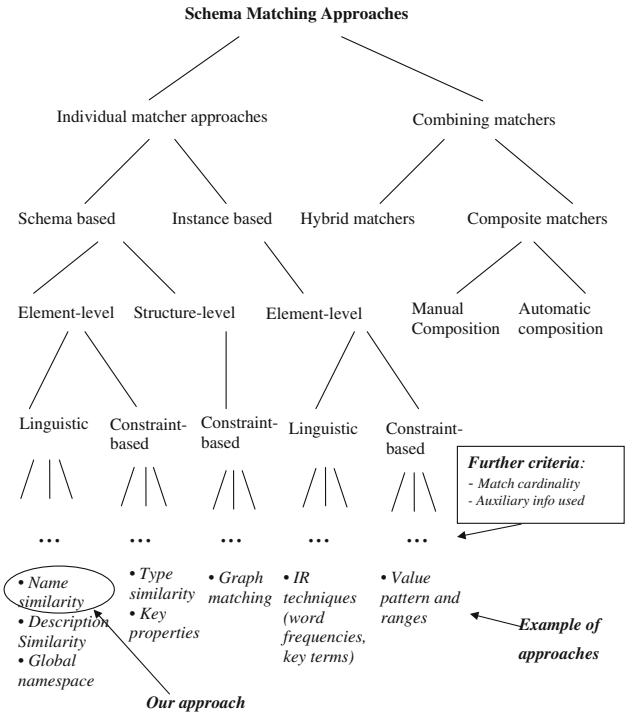

Fig. 10 Classification of schema matching approaches [49]

4.1.1 Linguistic approaches

Linguistic matchers use element names and text (i.e., words/tokens or sentences) to find semantically similar schema elements. We discuss here two schema-level approaches, name matching and description matching.

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.* An important sub-case is the equality of names from the same XML namespace, since this ensures that the same names indeed bear the same semantics.
- *Equality of canonical name representations after stemming and other preprocessing.* This is important to deal with special prefix/suffix symbols (e.g., CName → customer name, and EmpNO → employee number)
- *Equality of synonyms.* This is important to deal with synonyms (e.g., *car* synonymously matches *automobile*).
- *Equality of hypernyms.* Hypernym is a word that is more generic than a given word. For example, *fruit* is a hypernym of *apple* or *orange*. So, *apple* is–a *fruit* and *orange* is–a *fruit* imply: *apple* hypernymically matches *fruit*, *orange* hypernymically matches *fruit*, and *apple* matches *orange* as siblings.

- *Similarity of names based on longest common substrings (LCS), edit distance, pronunciation, soundex.* (a phonetic algorithm for indexing names by their sound when pronounced in English and the basic aim is for names with the same pronunciation to be encoded into the same string so that matching can occur despite minor differences in spelling.), etc. [3]. For example, *representedBy* matches *representative* based on common substrings. We experimented with these measures (edit, soundex, LCS, etc.) thanks to a Perl package, produced by Grzegorz Kondrak from University of Alberta [30].

Solving any task related to synonyms and hypernyms normally requires the use of thesauri or dictionaries. General natural language dictionaries such as LDOCE,[9] Wordnet[10] may be useful; perhaps even multi-language dictionaries (e.g., English–German or German–English) to deal with input schemas in different languages. In addition, name matching can use domain- or enterprise-specific dictionaries and *is–a* taxonomies containing common names, synonyms and descriptions of schema elements, abbreviations, etc. These specific dictionaries require a substantial effort to be built up in a consistent way. The effort may be worth the investment, especially for schemas with relatively flat structure where dictionaries provide the most valuable matching hints. But corpus-based methods could be a better choice than dictionary-based methods as a balanced corpus covers a huge collection of both domain-dependent and domain-independent words including special terms and proper nouns. Furthermore, tools are needed to enable names to be accessed and (re-)used, such as within a schema editor when defining new schemas.

Homonyms (one of two or more words that have the same sound and often the same spelling but differ in meaning) can mislead a matching algorithm as homonyms are similar names that refer to different elements. Clearly, homonyms may be a part of natural language, such as *bank* (embankment, river bank) and *bank* (place where money is kept). A name matcher can reduce the number of wrongly matched candidates using mismatch information supplied by users or dictionaries, though it requires a substantial effort or at least, the matcher can offer a warning of the potential ambiguity due to multiple meanings of the name.

Name-based matching can identify multiple relevant matches for a given schema element; that is, it is not limited to finding just 1:1 matches. For example, it can match "address" with both "home address" and "office address". In the case of synonyms and hypernyms, the join-like processing involves a list D of word pairs and their similarity as a further input. Assume a relation-like representation as follows:

$S_1(\text{name}_{11}, \text{name}_{12}...)$

$S_2(\text{name}_{21}, \text{name}_{22}...)$

$D(\text{name}_{11}, \text{name}_{21}, \text{similarity})$ where similarity is a similarity score for $[\text{name}_{11}, \text{name}_{21}]$ between 0..1. Then a list of all match candidates can be generated by the following three-way join operation:

Select S_1.name, S_2.name, D.similarity
From S_1, S_2, D
where (S_1.name = D.name$_{11}$) and

(D.name$_{21}$ = S_2.name) and

(D.similarity > threshold)

The constraint here is that D will have to contain all relevant pairs of the transitive closure over similar names. For instance, if $\text{sim}(A, B) = 0.6$ and $\text{sim}(B, C) = 0.7$ are in D, then probabilistically we would expect D also to contain $\text{sim}(B, A) = 0.6$, $\text{sim}(C, B) = 0.7$ and $\text{sim}(A, C) = x$, $\text{sim}(C, A) = x$. Probabilistically, we would expect the similarity value x to be $0.6 \times 0.7 = 0.42$, but this depends on some factors such as the type of similarity, the use of homonyms, and perhaps other factors. For example, we might have sim(*deliver*, *ship*) = 0. 9 and sim(*ship*, *boat*) = 0.9, but not sim(*deliver*, *boat*) = x for a high similarity value x. Bright et al. [6] discuss another approach to assigning different weights to different types of similarity relationships.

4.1.2 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:

S_1 : empn//*employee name*

S_2 : 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 [44].

4.2 Proposed Name-based element-level schema matching method

We use *longest common subsequence* (LCS) [1] measure with some normalization and small modification for our string

[9] http://www.longman.com/ldoce.

[10] http://wordnet.princeton.edu.

Fig. 11 Maximal consecutive LCS starting at character 1

Algorithm $MCLCS_1$

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 earlier.

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** $r_i \in S_j$; that is, $S_j \bigcap r_i = r_i$
4. **return** r_i
5. **else** $r_i \leftarrow r_i \setminus c_\tau$; that is, 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

Fig. 12 Maximal consecutive LCS starting at any character n

Algorithm $MCLCS_n$

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 earlier.

1. **while** $|r_i| \geq \zeta$ // we usually set ζ to 1. Details are discussed in next section.
2. determine all n-gram from r_i where $n = |r_i|$ and $\overline{r_i}$ is the set of n-gram
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
4. // Max ($\overline{r_i}$) returns the maximum length n-gram from $\overline{r_i}$
5. **return** x
6. **else** $\overline{r_i} \leftarrow \overline{r_i} \setminus x$ // remove element x from set $\overline{r_i}$
7. **end if**
8. **end while**

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

similarity measures. We use three different modified versions of LCS and then take weighted sum of these[11]. Kondrak [30] showed that edit distance and the length of the longest common subsequence are special cases of n-gram distance and similarity, respectively. Melamed [40] 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 the length of the smaller string which sometimes has a significant value in similarity score.

We normalize the LCS so that it takes into account the length of both the smaller and the larger strings 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 be consecutive, but in database schema matching, consecutive common subsequence is important for high degree of matching. We use *maximal consecutive longest common subsequence* starting at character 1, $MCLCS_1$ (Fig. 11) and *maximal consecutive longest common subsequence* starting at any character n, $MCLCS_n$ (Fig. 12). In Fig. 11, 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 larger string, where matching must be from first character (character 1) for both of the strings. In Fig. 12, we present another algorithm that

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

takes two strings as input and returns the smaller string or maximal consecutive portions of the smaller string that consecutively match with the larger string, where matching may start from any character (character n) for both of the strings. We normalize $MCLCS_1$ and $MCLCS_n$ and call it *normalized* $MCLCS_1 (NMCLCS_1)$ and *normalized* $MCLCS_n$ $(NMCLCS_n)$, respectively.

$$v_2 = NMCLCS_1(r_i, s_j) = \frac{\{length(MCLCS_1(r_i, s_j))\}^2}{length(r_i) \times length(s_j)}$$

$$v_3 = NMCLCS_n(r_i, s_j) = \frac{\{length(MCLCS_n(r_i, s_j))\}^2}{length(r_i) \times length(s_j)}$$

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:

$$\alpha = w_1 v_1 + w_2 v_2 + w_3 v_3.$$

We set equal weights for our experiments. Theoretically, $v_3 \geq v_2$.

For example, if $r_i = albastru$ and $s_j = alabaster$, then

$$LCS(r_i, s_j) = albastr$$
$$MCLCS_1(r_i, s_j) = al$$
$$MCLCS_n(r_i, s_j) = bast$$
$$NLCS(r_i, s_j) = 7^2/(8 \times 9) = 0.68$$
$$NMCLCS + 1 = 2^2/(8 \times 9) = 0.056$$
$$NMCLCS_n(r_i, s_j) = 4^2/(8 \times 9) = 0.22$$
$$\text{String similarity, } \alpha = w_1 v_1 + w_2 v_2 + w_3 v_3$$
$$= 0.33 \times 0.68 + 0.33 \times 0.056 + 0.33 \times 0.22 = 0.32$$

We then use word similarity measure, normalize it (Fig. 13) and combine it with the string similarity to obtain a 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 as *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 each of these segmented words to our 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_is (say, δ) for which $r_i = s_j$, for all $r \in R_i$ and for all $s \in S_i$; that is, 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; that is, $r_i = \{c_1 c_2 \ldots c_\tau\}$ and any token $s_j \in S_j$ has η characters; that is, $s_j = \{c_1 c_2 \ldots c_\eta\}$where $\tau \leq \eta$. In other words, η is the length of the larger token and τ is the length of the smaller

Fig. 13 Similarity matching

Algorithm *similarityMatching*

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

1. $v \leftarrow$ ***SOCPMI***(r_i, s_j) // This method determines semantic similarity
2. // between two words. Any other similarity
3. // method can also be used instead.
4. **if** $v > \lambda$ // We discuss about λ in next section.
5. $v \leftarrow 1$
6. **else** $v \leftarrow v / \lambda$
7. **end if**

Output: v // v is the semantic similarity value between 0 and 1, inclusively

token. We calculate the followings:

$v_1 \leftarrow NLCS(r_i, s_j)$
$v_2 \leftarrow NMCLCS_1(r_i, s_j)$
$v_3 \leftarrow NMCLCS_n(r_i, s_j)$
$\alpha_{ij} \leftarrow w_1 v_1 + w_2 v_2 + w_3 v_3$;

α_{ij} is a weighted sum of v_1, v_2 and v_3 where w_1, w_2, w_3 are weights and $w_1 + w_2 + w_3 = 1$. We set equal weights for our experiments.

We put α_{ij} in row i and column j position of the matrix for all $i = 1 \ldots m - \delta$ and $j = 1 \ldots n - \delta$.

$$M_1 = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{1j} & \alpha_{1(n-\delta)} \\ \alpha_{21} & \alpha_{22} & \alpha_{2j} & \alpha_{2(n-\delta)} \\ \alpha_{i1} & \alpha_{i2} & \alpha_{ij} & \alpha_{i(n-\delta)} \\ \alpha_{(m-\delta)1} & \alpha_{(m-\delta)2} & \alpha_{(m-\delta)j} & \alpha_{(m-\delta)(n-\delta)} \end{bmatrix}$$

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 $\beta_{ij} (\beta_{ij} \leftarrow similarityMatching(r_i, s_j)$ (Fig. 13)) in row i and column j position of the matrix for all $i = 1, \ldots, m - \delta$ and $j = 1, \ldots, n - \delta$.

$$M_2 = \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{1j} & \beta_{1(n-\delta)} \\ \beta_{21} & \beta_{22} & \beta_{2j} & \beta_{2(n-\delta)} \\ \beta_{i1} & \beta_{i2} & \beta_{ij} & \beta_{i(n-\delta)} \\ \beta_{(m-\delta)1} & \beta_{(m-\delta)2} & \beta_{(m-\delta)j} & \beta_{(m-\delta)(n-\delta)} \end{bmatrix}$$

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 the factors to 0.5 means we consider them equally important.

$$M = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{1j} & \gamma_{1(n-\delta)} \\ \gamma_{21} & \gamma_{22} & \gamma_{2j} & \gamma_{2(n-\delta)} \\ \gamma_{i1} & \gamma_{i2} & \gamma_{ij} & \gamma_{i(n-\delta)} \\ \gamma_{(m-\delta)1} & \gamma_{(m-\delta)2} & \gamma_{(m-\delta)j} & \gamma_{(m-\delta)(n-\delta)} \end{bmatrix}$$

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 $\rho \leftarrow \rho \mathrm{U} \gamma_{ij}$) if $\gamma_{ij} \geq \varsigma$ (we will discuss about the similarity threshold, ς in the next section). We remove all the matrix elements of ith row and jth 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.

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

4.2.1 Choosing the values of ζ, λ and ς

The parameter 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 usually set ζ to 1. If we use 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 $MCLCS_1$. When we use $MCLCS_n$, 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 $MCLCS_1$ and $MCLCS_n$. If we set ζ to 3, the pair *min similarity* returns Ø for both $MCLCS_1$ and $MCLCS_n$. Basically, λ is dependent on the semantic similarity method we use. We choose the value of λ based on the maximum range of the similarity values for the semantic similarity method we use. We usually set λ to 20 when we use *SOCPMI* semantic similarity method because our experiments showed that 20 is the maximum of the region of values for best matches. We can use any other similarity measure including a dictionary-based or a hybrid approach. For example, if we use the Roget-based measure [27] than we need to set λ to 16. One of the main advantages of using distributional measures based on corpus is that it covers significantly more tokens than any dictionary-based measure. Theoretically, ς could be any value between 0 and 1 exclusively, 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 result, thus it is better to omit those.

4.3 Walk-through examples

We provide two examples that describe the proposed method and determine the similarity score. In example 1, we use two real element names from a database schema and in example 2, we use two element names that we created, in order to better illustrate the method (to cover all its strength at once).

4.3.1 Example 1

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 larger 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 larger token. So, $v_1 = v_2 = v_3 = 0$ and $\alpha_{11} = 0$.

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

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

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

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

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

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 ith ($i = 1$) row and jth ($j = 1$) column. We proceed to next step as $m - \delta - |\rho| = 0$. (Here, $m = 2, \delta = 1$ and $|\rho| = 1$)
Step 6:

$$\begin{aligned} Similarity\ Score(R_i, S_j) &= \frac{\left(\delta + \sum_{i=1}^{|\rho|} \rho_i\right) \times (m+n)}{2mn} \\ &= (1 + 0.163) \times 4/8 \\ &= 0.582 \end{aligned}$$

4.3.2 Example 2

Let
R_i = "allmileage_make_maxkm",
S_j = "make_minmile_distance_possible_take".

Step 1: After eliminating all special characters and punctuations, if any, and then using word segmentation method and lemmatizing, we get $R_i = \{all, mileage, make, max, km\}$ and $S_j = \{make, min, mile, distance, possible, take\}$ where $m = 5$ and $n = 6$.

Step 2: Only one token (i.e., *make*) in R_i exactly matches with S_j therefore we set δ to 1. We remove *make* from both R_i and S_j. So, $R_i = \{all, mileage, max, km\}$ and $S_j = \{min, mile, distance, possible, take\}$. As $m - \delta \neq 0$, we proceed to next step.
Step 3: We construct a 4×5 *matching matrix*, M_1. Consider the *mileage possible* pair where *length*(*LCS*(*mileage*, *possible*)) = 3, $\eta = 8$ is the length of the larger token (*possible*), $\tau = 7$ is the length of the smaller token (*mileage*) and 2 is the maximal length of the consecutive portions of the smaller token that consecutively match with the larger token, where matching starts from third character of the smaller token and seventh character of the larger token. So, $v_1 = 3^2/(8 \times 7) = 0.16$

$v_2 = 0$
$v_3 = 2^2/(8 \times 7) = 0.071$
and $\alpha_{24} = 0.33 \times v_1 + 0.33 \times v_2 + 0.33 \times v_3 = 0.076$

$$M_1 = \begin{array}{c} \\ all \\ mileage \\ max \\ km \end{array} \begin{array}{c} \begin{array}{ccccc} min & mile & distance & possible & take \end{array} \\ \begin{bmatrix} 0 & 0.055 & 0.041 & 0.027 & 0.082 \\ 0.188 & 0.565 & 0.058 & 0.076 & 0.058 \\ 0.11 & 0.082 & 0.027 & 0 & 0.055 \\ 0.11 & 0.082 & 0 & 0 & 0.123 \end{bmatrix} \end{array}$$

Step 4: We construct a 4×5 *similarity matrix*, M_2. Here, $\lambda = 20$ as we used *SOCPMI* method.

$$M_2 = \begin{array}{c} \\ all \\ mileage \\ max \\ km \end{array} \begin{array}{c} \begin{array}{ccccc} min & mile & distance & possible & take \end{array} \\ \begin{bmatrix} 0.172 & 0.233 & 0.48 & 0 & 0.813 \\ 0.587 & 0.976 & 0.826 & 0 & 0.558 \\ 0.199 & 0.194 & 0.141 & 0 & 0.243 \\ 0.67 & 0.962 & 0.89 & 0 & 0.408 \end{bmatrix} \end{array}$$

Step 5: We construct a 4×5 *joint matrix*, M and assign equal weight factor by setting both ψ and φ to 0.5.

$$M = \begin{array}{c} \\ all \\ mileage \\ max \\ km \end{array} \begin{array}{c} \begin{array}{ccccc} min & mile & distance & possible & take \end{array} \\ \begin{bmatrix} 0.086 & 0.144 & 0.26 & 0.013 & 0.447 \\ 0.388 & 0.771 & 0.442 & 0.038 & 0.308 \\ 0.154 & 0.138 & 0.084 & 0 & 0.149 \\ 0.39 & 0.522 & 0.445 & 0 & 0.266 \end{bmatrix} \end{array}$$

We find the maximum-valued matrix-element, $\gamma_{ij} = 0.771$ and add it to ρ as $\gamma_{ij} \geq \varsigma$ (we use $\varsigma = 0.01$ in this example). So, $\rho = \{0.771\}$. The new M after removing ith ($i = 2$) row and jth ($j = 2$) column is:

$$M = \begin{array}{c} \\ all \\ max \\ km \end{array} \begin{array}{c} \begin{array}{cccc} min & distance & possible & take \end{array} \\ \begin{bmatrix} 0.086 & 0.26 & 0.013 & 0.447 \\ 0.154 & 0.084 & 0 & 0.149 \\ 0.39 & 0.445 & 0 & 0.266 \end{bmatrix} \end{array}$$

We find the maximum-valued matrix-element, $\gamma_{ij} = 0.447$ for this new M and add it to ρ as $\gamma_{ij} \geq \varsigma$. So, $\rho = \{0.771, 0.447\}$. The new M after removing ith ($i = 1$) row and jth ($j = 4$) column is:

$$M = \begin{array}{c} \\ max \\ km \end{array} \begin{array}{c} \begin{array}{ccc} min & distance & possible \end{array} \\ \begin{bmatrix} 0.154 & 0.084 & 0 \\ 0.39 & 0.445 & 0 \end{bmatrix} \end{array}$$

Here, 0.445 is the maximum-valued matrix-element and $\gamma_{ij} \geq \varsigma$. So, $\rho = \{0.771, 0.447, 0.445\}$. The new M after removing ith ($i = 2$) row and jth ($j = 2$) column is:

$$M = \begin{array}{c} \\ max \end{array} \begin{array}{c} \begin{array}{cc} min & possible \end{array} \\ \begin{bmatrix} 0.154 & 0 \end{bmatrix} \end{array}$$

We find 0.154 as the maximum-valued matrix-element and $\gamma_{ij} \geq \varsigma$. So, $\rho = \{0.771, 0.447, 0.445, 0.154\}$. The new M is empty after removing ith ($i = 1$) row and jth ($j = 1$) column.

We proceed to next step as $m - \delta - |\rho| = 0$. (Here, $m = 5, \delta = 1$ and $|\rho| = 4$)

Step 6:

$$\begin{aligned} Similarity\ Score(R_i, S_j) &= \frac{\left(\delta + \sum_{i=1}^{|\rho|} \rho_i\right) \times (m + n)}{2mn} \\ &= (1 + 1.817) \times 11/60 \\ &= 0.516 \end{aligned}$$

4.4 Evaluation and experimental 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. [39], 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. [39], all of it, while they used 75% of it, randomly selected. We could not reproduce the exact 75% that they used. Figures 14 and 15 are two sample schemas from *auto* domain (vname are the element names to be matched), while Fig. 16 is their manual mapping (the tags <left> and <right> are used to show an element name from the first schema that matches with an element name from the second schema).

In each domain, they manually created mappings between randomly chosen schema pairs. The matches were *one–many*; that is, 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 2 provides detailed information about each of the two domains and our results.

In 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 11 different similarity thresholds ranging from 0 to 1 with interval 0.1, e.g., using *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. Precision vs. similarity threshold curves and recall vs. similarity threshold curves of the two web domains for the 11 different similarity thresholds are shown is Figs. 17 and 18, respectively. P–R curves of the two web domains for the 11 different similarity thresholds are shown in Fig. 19 where the similarity threshold decreases from left to right in the figure. Figure 20 shows F-measure vs. similarity threshold curves; it is obvious that a lower similarity threshold ($\approx$ 0.2) gives a better 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 semantic word similarity measures. If two strings have perfect semantic word similarity score (≈ 1) and no string similarity score (≈ 0), it is practically a perfect matching (e.g., *car* and *vehicle*); this lowers the total similarity score. Again, we multiply this total score by the reciprocal harmonic mean of m and n to obtain a balance similarity score which also lowers the final similarity value.

In Fig. 18, when we use string similarity threshold score of 1 (i.e., matching the element names exactly, therefore no semantic similarity matching is needed), we obtain recall values of 0.133 and 0.107 for *auto* and *real estate* domains, respectively. We can consider these scores as the baselines.

Madhavan et al. [39] 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

Fig. 14 A sample schema named "**401*carfinder***" from *auto* domain

```
<domain name="Automobile">
<search>
<source name="401carfinder"
url="http://www.401carfinder.com/ad_search_form.htm" />
<element>
<element vname="Region" text="Region">
<selectbox>
  <item>Alberta</item>
  <item>British Columbia</item>
  <item>Nova Scotia</item>
  <item>Ontario: Windsor</item>
  <item>Quebec</item>
  <item>USA</item>
</selectbox>
</element>
<element vname="MAKE" text="Make">
<selectbox>
  <item>All Makes</item>
  <item>Acura</item>
  <item>BMW</item>
  <item>Cadillac</item>
  <item>Yamaha</item>
</selectbox>
</element>
<element vname="Model" text="Model (optional)">
  <textbox />
</element>
<element vname="MinYear" text="Year (optional)">
   <textbox />
</element>
<element vname="MaxYear" text="to">
   <textbox />
</element>
<element vname="MaxPrice" text="Max. Price (optional)">
   <textbox />
</element>
<element vname="Keyword" text="Keyword (optional)">
    <textbox />
</element>
<element vname="Sort" text="sort results by">
<RadioButton>
  <item>price</item>
  <item>year</item>
  <item>make</item>
</RadioButton>
 </element>
</element>
</search>
</domain>
```

Fig. 15 A sample schema named "***AutoWeb***" from *auto* domain

```
<domain name="Automobile">
<search>
<source name="AutoWeb"
url="http://www.autoweb.com/content/buy/used/index.cfm" />
<element>
<element vname="vehicle" text="Select Vehicle">
<selectbox>
  <item>Acura CL</item>
  <item>Volvo 240</item>
  <item>Yugo Cabrio</item>
</selectbox>
</element>
<element vname="search_mileage_int" text="search within">
<selectbox>
  <item>25</item>
  <item>50</item>
  <item>100</item>
</selectbox>
</element>
<element vname="Entered_Postal_Code_vch" text="miles of">
  <textbox />
</element>
<element vname="Low_Price" text="Price Range">
<selectbox>
  <item>Any</item>
  <item>$5,000</item>
  <item>$15,000</item>
  <item>$20,000</item>
  </selectbox>
</element>
<element vname="High_Price" text="to">
<selectbox>
  <item>Any</item>
  <item>$25,000</item>
  <item>$40,000 +</item>
</selectbox>
</element>
<element vname="numCarsOnOnePage" text="Vehicles per page">
<selectbox>
  <item>5</item>
  <item>10</item>
  <item>15 </item>
</selectbox>
</element>
</element>
</search>
</domain>
```

Fig. 16 Manual mapping between "**401*carfinder***" and "***AutoWeb***"

```
<mappings schema1="401carfinder" schema2="AutoWeb">
<match>
<left>
        <item>Region</item>
</left>
<right>
        <item>Entered_Postal_Code_vch</item>
</right>
</match>
<match>
<left>
        <item>MAKE</item>
</left>
<right>
        <item>vehicle</item>
</right>
</match>
<match>
<left>
        <item>Model</item>
</left>
<right>
        <item>vehicle</item>
</right>
</match>
<match>
<left>
        <item>MaxPrice</item>
</left>
<right>
        <item>High_Price</item>
</right>
</match>
</mappings>
```

positive and negative examples for the element on which it is being trained. The *direct* method uses the same base learners of *augment* method, but the training data for these learners are extracted only from the schemas being matched. *Pivot* is the method that computes cosine distance of the interpretation vectors of the two elements directly.

In Fig. 21, the *direct*, *pivot* and *augment* methods for the *auto* domain 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.

In Fig. 22, the *direct*, *pivot* and *augment* methods for the *real estate* domain achieved precision of around 0.78, 0.71 and 0.76, recall of around 0.69, 0.74, 0.81 and F-measure of around 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 0.2 as similarity threshold.

Generally, it seems that precision does matter 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 verification procedure then recall is as important as precision; that is, F-measure does matter more than precision.

Our method is computationally less intensive than the method of Madhavan et al. [39] because it uses a single property in matching (element names), while their method uses multiple properties (names, descriptions, instance values,

Table 2 Characteristics of the evaluation domains and our results

Domain name	Number of schemas	Number of manual mappings	Similarity threshold score in our method	Number of predicted mapping pairs	Number of correct mapping pairs	Number of manually created mapping pairs
Auto	30	95	0	33,116	769	769
			0.1	961	628	
			0.2	769	596	
			0.3	701	564	
			0.4	689	558	
			0.5	642	530	
			0.6	501	424	
			0.7	438	382	
			0.8	200	192	
			0.9	176	176	
			1.0	103	103	
Real estate	20	57	0	4,262	280	280
			0.1	364	232	
			0.2	310	211	
			0.3	248	176	
			0.4	228	173	
			0.5	203	164	
			0.6	155	130	
			0.7	124	105	
			0.8	59	55	
			0.9	48	48	
			1.0	30	30	

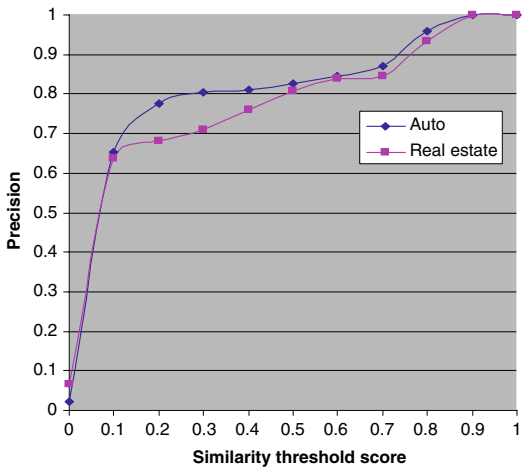

Fig. 17 Precision vs. similarity threshold curves of the two web domains for 11 different similarity threshold

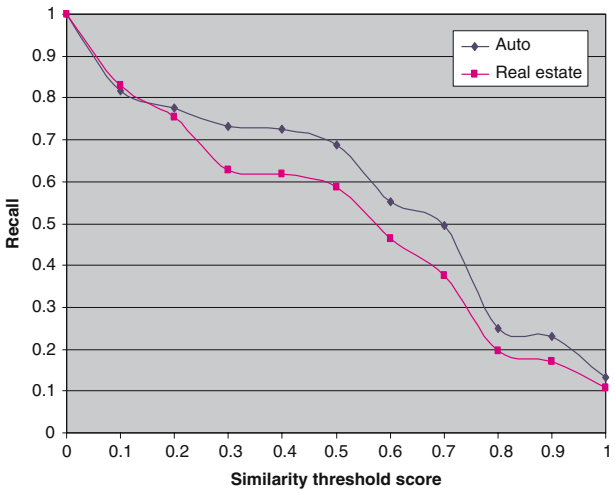

Fig. 18 Recall vs. similarity threshold curves of the two web domains for 11 different similarity threshold

context). We feel that a rigorous comparison is not possible, since the algorithm of Madhavan et al. is not described in sufficient details. However, we believe that the complexity of our matcher is similar to that of Madhavan et al.'s name learner; in addition they include the complexity of three other learners.

Finally, we wanted to measure the contribution of our two new methods, the semantic similarity method and the text segmentation method, to the task of database schema matching.

When we used Lin's [38] *WordNet*-based word similarity method instead of our corpus-based word similarity method,

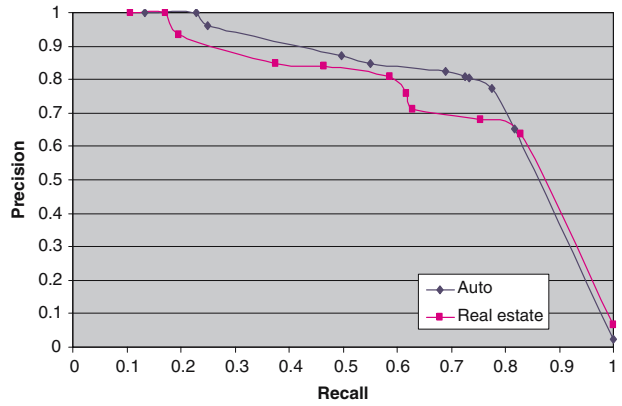

Fig. 19 *P–R* curves of the two web domains for 11 different similarity threshold (similarity threshold decreases from *left* to *right*)

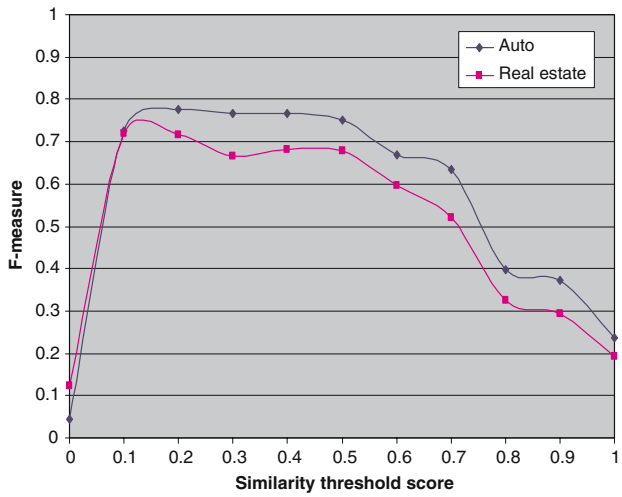

Fig. 20 F-measure vs. similarity threshold curves of the two web domains for 11 different similarity threshold

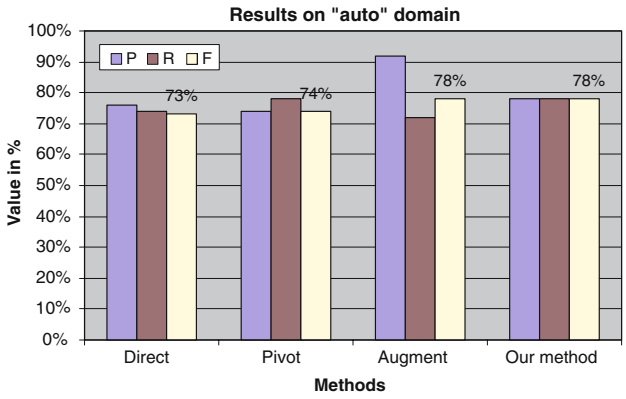

Fig. 21 Results on the *auto* domain

for the *auto* domain, we achieved a precision of 0.71, recall of 0.63, and F-measure of 0.66. It matched 680 elements, out of which 485 elements were among the 769 manually matched elements. We compare this to the results of our best run for the *auto* domain that had F-measure of 0.78

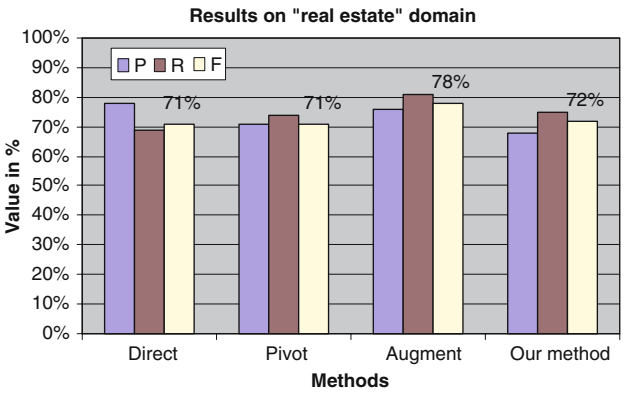

Fig. 22 Results on the *real estate* domain

(with precision and recall of 0.78 as well).[12] The decrease in F-measure due to the replacement of our semantic similarity method is 0.12; the decrease in precision and recall is 0.07 and 0.15, respectively.

When we used a simplistic segmentation approach (segmentation using only capitalization and punctuation) instead of our proposed word segmentation approach, for the *auto* domain, we achieved a precision of 0.76, recall of 0.68, and F-measure of 0.71. It matched 687 elements, out of which 521 elements were among the 769 manually matched elements. The loss in F-measure, precision and recall due to the simplistic segmentation method is 0.08, 0.02 and 0.10, respectively.

We can conclude that our semantic similarity method contributed a significant improvement (12 percentage points for the *auto* domain). Our word segmentation method also contributed significantly (an improvement of 8 percentage points).

5 Conclusion and future work

5.1 Conclusion

In this paper, we addressed the task of database schema matching. First, we evaluated a new corpus-based word similarity measure, called SOC-PMI and compared it with existing word similarity measures. We performed intrinsic evaluation on the noun pairs mentioned earlier. We also performed a task-based evaluation: solving synonyms test questions. One of the main characteristics of the SOC-PMI method is that we can determine the semantic similarity of two words even though they do not co-occur within the window size at all in the corpus. Actually, we are considering the second-order co-occurrences, as we are judging also by the co-occurrences of the neighbor words, not only the co-occurrence of the two

[12] We used 0.2 as the similarity threshold, to have the same threshold for the compared systems.

target words. This is not the case for PMI-IR and many other corpus-based semantic similarity measures.

Second, we proposed a word segmentation method that could be exploited in web search engine to provide better suggestion when the search text contains three or more words in the "desegmented" part. The method can also effectively distill new words, special terms and proper nouns when the corpus covers a huge collection of both domain-dependent and domain-independent words, and it can effectively avoid statistical errors in shorter strings which belong to a longer one. Experimental results show that our word segmentation method can segment words with high precision and high recall.

Finally, we exploit both the semantic similarity and the word segmentation method in our proposed name-based element-level schema matching method.

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 P2P data management or online query processing in P2P environment. Our method is scalable, in the sense that, if needed, we could also add other properties (i.e., text description, data instance matching) to obtain a better schema matching result.

5.2 Future work

We plan to apply our proposed second-order co-occurrence PMI method to other tasks, such as measuring the semantic similarity of two texts and detecting semantic outliers in speech recognition transcripts. The SOC-PMI method may also be helpful as a tool to aid in the automatic construction of the synonyms of a word. A very naïve approach would be as follows. First, we need to sort out the significant words list based on PMI values for the word (say, x) we are interested to find the synonyms. If there are n significant words in this words list, we will apply the SOC-PMI method for each possible pair mapping from x to n. Instead of taking the similarity value, we will consider all the second-order co-occurrence types and sort out this types list based on PMI values. The words on the top of the list could be the best candidates for synonyms of the word.

Our corpus-based word segmentation method can be extended as a hybrid method with some additions to the algorithms. The absence of type frequencies in dictionaries means that we can only use the length of the types. In that case, we need to focus on what type to choose for the same length types that share some common characters. Again, we cannot choose whether we take the elements of *leftMaxMatching* or *rightMaxMatching* when both of them return the same number of elements as we cannot use entropy rate for the absence of type frequencies. Future directions also involve integrating the current word segmentation algorithm into a larger system for comprehensive and context-based word analysis.

Our proposed schema matching method together with the semantic similarity of words method can be further extended for the task of paraphrase recognition, entailment identification and measuring the semantic similarity of texts. A corpus-based measure is useful to identify any similarity between words like *President* and *Clinton* from sentences '*Mr. President was supposed to visit Europe*' and '*Mr. Clinton was supposed to visit Europe*'. The proposed schema matching method can also be updated in exactly the same way as the text similarity approach to exploit text description matching, another approach for schema matching.

Incorporating equality of canonical name representations for special prefix/suffix symbols (e.g., CName → customer name, and EmpNO → employee number) will enhance the performance of the method. We tried with the Opaui,[13] a collection of lists of acronyms, abbreviations, and initialisms on the Word Wide Web, which has 353,494 entries from 128 different categories, but it even made the result worse. The reason is that misleading acronyms, abbreviations or initialisms return lower string similarity and semantic word similarity scores.

Our name-based schema matching method can be augmented with rule-based methods to improve the accuracy in domain-specific schema matching.

Notice that our algorithms assumed that most element names are tokenizable (contain words or fragments of words), 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. For example, in the Real Estate domain, a schema named "CommercialRealEstate" had five fields/elements: cata, beds, catb, catc, state, and another schema named "RealyInvestor-1" had seven fields/elements: OptionListSelectedTypes, tScMinPriceSale, tScMaxPriceSale, tScMinSfSale, tScMaxSfSale, tScMinUnits, tScMaxUnits. Their manual matches are as follows: (cata = tScMinPriceSale and tScMaxPriceSale), (catb = tScMinSfSale and tScMaxSfSale), (catc = OptionListSelectedTypes). 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 these type of data are not very frequent in real-world web data sources. To test this hypothesis further, we collected 112 element names from 12 websites.[14] Among them only eight names were not tokenizable.

[13] http://www.abbreviationz.com/.

[14] The list is available at http://www.site.uottawa.ca/~diana/elements.htm.

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. To quickly test this idea, we implemented a simple type instance matcher that verifies the type of instance values. In case our name matcher decided to match two fields, we did not accept the match if the fields had different types, for example if one field was a string and the other was numeric. In this way, for the *auto* domain, we eliminated 52 incorrect matches; increasing the precision from 0.78 to 0.83 and the F-measure from 0.78 to 0.80. The recall stayed the same because all the eliminated matches were indeed wrong matches. 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, if available. Our word-level similarity measure can be used to determine the similarity level of two texts.

References

1. Allison, L., Dix, T.I.: A bit-string longest-common-Subsequence algorithm. Inf. Process. Lett. **23**, 305–310 (1986)
2. Batini, C., Lenzerini, M., Navathe, S.B.: A comparative analysis of methodologies for database schema integration. ACM Comput. Surv. **18**(4), 323–364 (1986)
3. Bell, G.S., Sethi, A.: Matching records in a national medical patient index. Commun. ACM (CACM) **44**(9), 83–88 (2001)
4. Brent, M.: An efficient, probabilistically sound algorithm for segmentation and word discovery. Mach. Learning **34**, 71–106 (1999)
5. Brent, M., Cartwright, T.: Distributional regularity and phonotactics are useful for segmentation. Cognition **61**, 93–125 (1996)
6. Bright, M.W., Hurson, A.R., Pakzad, S.H.: Automated resolution of semantic heterogeneity in multi databases. Trans. Database Systems (TODS) **19**(2), 212–253 (1994)
7. Brill, E.: Some advances in transformation-based part of speech tagging. In Proceedings of the Twelfth National Conference on Artificial Intelligence, pp. 748–753. AAI Press/MIT Press (1994)
8. Brown, P.F., DeSouza, P.V., Mercer, R.L., Watson, T.J., Della Pietra, V.J., Lai, J.C.: Class-based n-gram models of natural language. Comput. Linguist. **18**, 467–479 (1992)
9. Budanitsky, A., Hirst, G.: Evaluating WordNet-based measures of semantic distance. Comput. Linguist. **32**(1) (2006)
10. Buckley, C., Salton, J.A., Singhal, A.: Automatic query expansion using Smart: TREC 3. In Proceedings of the Third Text Retrieval Conference, Gaithersburg (1995)
11. Christiansen, M., Allen, J.: Coping with Variation in Speech Segmentation. In: Proceedings of GALA 1997: Language Acquisition: Knowledge Representation and Processing, pp. 327–332 (1997)
12. Christiansen, M., Allen, J., Seidenberg, M.: Learning to segment speech using multiple cues: a connectionist model. Language Cogn. Process. **13**, 221–268 (1998)
13. Church, K.W., Hanks, P.: Word association norms, mutual information, and lexicography. Comput. Linguist. **16**(1), 22–29 (1990)
14. Daelamans, W., van den Bosch, A., Weijters, A.: IGTree: Using trees for compression and classification in lazy learning algorithms. Artif. Intell. Rev. **11**, 407–423 (1997)
15. Dagan, I., Lee, L., Pereira, F.C.N.: Similarity based models of word cooccurrence probabilities. Mach. Learning **34**(1–3), 43–69 (1999)
16. Dale, R., Moisl, H., Somers, H.: Handbook of Natural Language Processing. Marcel Dekker, Inc., New York (2000)
17. Deligne, S., Bimbot, F.: Language modeling by variable length sequences: theoretical formulation and evaluation of multigrams. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP-95) (1995)
18. de Marcken, C.: The unsupervised acquisition of a lexicon from continuous speech. Technical Report AI Memo No. 1558, M.I.T., Cambridge, MA (1995)
19. Do, H.H., Rahm, E.: COMA—a system for flexible combination of schema matching approaches. In: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), pp. 610–621 (2002)
20. Dunning, T.: Accurate methods for the statistics of surprise and coincidence. Comput. Linguist. **19**, 61–74 (1993)
21. Fung, P., Wu, D.: Improving Chinise tokenization with linguistic filters on statistical lexical acquisition. In: Fourth Conference Applied Natural Language Processing, Stuttgart, pp. 180–181 (1994)
22. Gao, J., Li, M., Wu, A., Huang, C.-N.: Chinese word segmentation and named entity recognition: a pragmatic approach. Comput. Linguist. **31**(4) (2005)
23. Grefenstette, G.: Automatic thesaurus generation from raw text using knowledge-poor techniques. In: Making sense of Words, 9th Annual Conference of the UW Centre for the New OED and Text Research (1993)
24. Grefenstette, G.: Finding semantic similarity in raw text: the deese antonyms. In: Goldman, R., Norvig, P., Charniak, E., Gale, B. (eds.) Working Notes of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language, pp. 61–65. AAAI Press (1992)
25. Hua, Y.: Unsupervised word induction using MDL criterion. In: Proceedings ISCSL 2000, Beijing (2000)
26. Inkpen, D., Désilets, A.: Semantic similarity for detecting recognition errors in automatic speech transcripts. In: Proceedings of Empirical Methods in Natural Language Processing (EMNLP 2005), Vancouver (2005)
27. Jarmasz, M., Szpakowicz, S.: Roget's thesaurus and semantic similarity. In: Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP-2003), Borovets, Bulgaria, pp. 212–219 (2003)
28. Kang, J., Naughton, J.F.: On schema matching with opaque column names and data values. In: Proceedings of Special Interest Group on Management of Data (SIGMOD 2003), San Diego, pp. 205–216 (2003)
29. Kit, C., Wilks, Y.: Unsupervised learning of word boundary with description length gain. In: Proceedings CoNLL99 ACL Workshop, Bergen (1999)
30. Kondrak, G.: N-gram similarity and distance. In: Proceedings of the Twelfth International Conference on String Processing and Information Retrieval (SPIRE 2005), Buenos Aires, pp. 115–126 (2005)
31. Landauer, T.K., Dumais, S.T.: A solution to plato's problem: the latent semantic analysis theory of the acquisition, induction, and representation of knowledge. Psychol. Rev. **104**(2), 211–240 (1997)
32. Lee, L.: Measures of distributional similarity. In: Proceedings of the Association for Computational Linguistics (ACL-1999), pp. 23–32 (1999)

33. Lesk, M.E.: Word-word associations in document retrieval systems. Am. Doc. **20**(1), 27–38 (1969)
34. Lesk, M.E.: Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In: Proceedings of the Conference of Special Interest Group on Design Of Communication (SIGDOC), Toronto (1986)
35. Li, H., Abe, N.: Word clustering and disambiguation based on co-occurrence data. In: Proceedings of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING-ACL), pp. 749–755 (1998)
36. Lin, C.Y., Hovy, E.H. (2003) Automatic evaluation of summaries using n-gram co-occurrence statistics. In: Proceedings of Human Language Technology Conference (HLT-NAACL 2003), Edmonton
37. Lin, D.: Automatic retrieval and clustering of similar words. In Proceedings of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING-ACL), pp. 768–774 (1998)
38. Lin, D.: An information-theoretic definition of similarity. In: Proceedings of the 15th International Conference on Machine Learning, pp. 296–304 (1998)
39. Madhavan, J., Bernstein, P., Doan, A., Halevy, A.: Corpus-based Schema Matching. In: International Conference on Data Engineering (ICDE-05), pp. 57–68 (2005)
40. Melamed, I.D.: Bitext maps and alignment via pattern recognition. Comput. Linguist. **25**(1), 107–130 (1999)
41. Mikheev, A.: Automatic rule induction for unknown word guessing. Comput. Linguist. **23**(3), 405–423 (1997)
42. Miller, G.A., Charles, W.G.: Contextual correlates of semantic similarity. Language Cogn. Process. **6**(1), 1–28 (1991)
43. Miller, G.A.: WordNet: a lexical database for English. Commun. ACM **38**(11), 39–41 (1995)
44. Milo, T., Zohar, S.: Using Schema matching to simplify heterogeneous data translation. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 122–133 (1998)
45. Pantel, P., Lin, D.: Discovering word senses from text. In: Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 613–619 (2002)
46. Pedersen, T., Patwardhan, S., Michelizzi, J.: WordNet: Similarity—measuring the relatedness of concepts. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04), July 25–29, San Jose (Intelligent Systems Demonstration)
47. Peng, F., Schuurmans, D.: A hierarchical EM approach to word segmentation. In: Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium (NLPRS 2001). pp. 475–480, Tokyo, Japan (2001)
48. Rabiner, L.: A tutorial on hidden markov models and selected applications in speech recognition. In: Proc. IEEE 77(2), 257–286 (1989)
49. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. Int. J. Very Large Data Bases (VLDB) **10**(4), 334–350 (2001)
50. Rao, C.R.: Diversity: Its measurement, decomposition, apportionment and analysis. Sankyha: Indian J. Statist. **44**(A), 1–22 (1983)
51. Resnik, P.: Using information content to evaluate semantic similarity. In Proceedings of 14th International Joint Conference on Artificial Intelligence, Montreal, pp. 448–453 (1995)
52. Rosenfeld, R.: A maximum entropy approach to adaptive statistical language modeling. Comput Speech Language **10**, 187–228 (1996)
53. Rubenstein, H., Goodenough, J.B.: Contextual correlates of synonymy. Commun. ACM **8**(10), 627–633 (1965)
54. Rumelhart, D.E., McClelland, J.: On learning the past tense of English verbs. In: Parallel Distributed Processing, vol. II, pp. 216–271, MIT Press, Cambridge, (1986)
55. Saffran, J.R., Newport, E.L., Aslin, R.N.: Word segmentation: The role of distributional cues. J. Memory Language **35**, 606–621 (1996)
56. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill (1983)
57. Seligman, L., Rosenthal, A., Lehner, P., Smith, A.: Data integration: Where does the time go? Bull. Tech. Comm. Data Eng. 25(3) (2002)
58. Shannon, C.E., Weaver, W.: The mathematical theory of communication. University of Illinois Press, Urbana (1963)
59. Sproat, R., Shih, C., Gale, W., Chang, N.: A stochastic finite-state word-segmentation algorithm for Chinese. Comput. Linguist. **22**(3), 377–404 (1996)
60. Sproat, R., Shih, C., Gale, W., Chang, N.: A stochastic word segmentation algorithm for a Mandarin text-to-speech system. In: 32nd Annual Meeting of the Association for Computational Linguistics pp. 66–72. Las Cruces (1994)
61. Turney, P.D.: Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In: Proceedings of the Twelfth European Conference on Machine Learning (ECML-2001), pp. 491–502 (2001)
62. Vechtomova, O., Robertson, S.: Integration of collocation statistics into the probabilistic retrieval model. In: 22nd Annual Colloquium on Information Retrieval Research, Cambridge (2000)
63. Weeds, J., Weir, D., McCarthy, D.: Characterising measures of lexical distributional similarity. In: Proceedings of the 20th International Conference on Computational Linguistics, COLING-2004 pp. 1015–1021. Geneva (2004)
64. Xu, J., Croft, B.: Improving the effectiveness of information retrieval. ACM Trans. Inf. Syst. **18**(1), 79–112 (2000)
65. Yarowsky, D.: Word-sense disambiguation using statistical models of Roget's categories trained on large corpora. In: Proceedings of the International Conference on Computational Linguistics (COLING-92), pp. 454–460, Nantes, (1992)