Comparison of Methods to Assess Similarity between Phrases*

Renzo Angles, Valeria Araya, Jesus Concha, and Rodrigo Paredes

Department of Computer Science, Universidad de Talca, Chile
{rangles,econcha,raparede}@utalca.cl, valeria.araya.m@gmail.com

Abstract. We study the problem of similarity between phrases. To do so, we study three similarity methods. The first one considers the commonalities and differences of the two phrases. The second one is an extension of the well-known Levenshtein-Damerau distance in a word oriented fashion. The third one considers the sequentiality of the phrases and is resistant to phrases with repeated words. Finally, we show an experimental evaluation of our methods in both English and Spanish corpora.

Keywords: Phrase similarity, Levenshtein Distance, Word oriented methods.

1 Introduction

A phrase is a sequence of consecutive words. The notion of phrase similarity has been studied in several application domains, including document retrieval [1], web document clustering [3,4] and automatic machine translations [5,10].

We study phrase similarity in the perspective of improving the quality of automatic translations. Our study is motivated by the deficiencies of current techniques to accurately translate a text from a source to a target language. For example, Table 1 shows the results of translating the phrase "the trip was busy" from English to Spanish, by using four translation services. Considering that the expected result is "el viaje estuvo ocupado", we see that none of them was able to provide the correct translation.

Refining a machine translation implies to modify the translation under a given criteria in order to improve its quality. In the above example, the translated phrase "el viaje fue ocupado" (returned by Google Translator) can be refined by replacing the word "fue" by "estuvo". Our approach for refining is based on exploiting the Web as a huge source of phrases. The idea is building a database of (likely) well written phrases obtained from the Web. Then, given a translated phrase, we will try to find a well written phrase in the database which can be used to refine the input phrase.

In order to solve this problem, we need to define a metric to measure the similarity between phrases. That is, given two phrases, the metric returns a score which determines how similar the two phrases are (the higher the score

* This work is partially funded by Fondecyt grant 1131044, Chile.

E. Bayro-Corrochano and E. Hancock (Eds.): CIARP 2014, LNCS 8827, pp. 255–263, 2014.

Table 1. Examples of translations, English-Spanish, for the phrase "the trip was busy"

Translation Service	Result
Google Translator (`translate.google.com/`)	el viaje fue ocupado
Bing Translator (`www.bing.com/translator`)	el viaje estaba ocupado
Babylon (`traductor.babylon.com/`)	el viaje estaba ocupado
El Mundo Traductor (`www.elmundo.es/traductor/`)	el viaje estaba ocupado

the more similarity). Hence, each phrase in the database can be compared with the input phrase and ranked according to its score. The phrase with the highest score is the solution to the problem.

Contributions. This paper presents new contributions to the problem of computing similarity between phrases. First, we formalize a basic notion of similarity found in the literature [8] which is based on the commonalities and differences between two phrases. Second, we extend the Levenshtein distance [7] to compare phrases by considering word-oriented edit operations. Third, we present a sequential method to calculate the cost of a phrase-to-phrase transformation which is simpler than Levenshteins's but preserves the precision. Four, we evaluate and compare the effectiveness and efficiency of the methods in terms of their precision, recall, coverage, sensitivity and execution time.

1.1 Related Work

The notion of similarity between phrases has been studied in several application domains, including machine translation [10,2] and phrase-based document similarity [3,4]. To best of our knowledge, there exist no work concerning the study and comparison of metrics to measure the similarity between phrases. The closest work is the one presented by Lin [8] in 1994. In this article, Lin presents an information-theoretic definition of similarity and demonstrate how to use it to measure similarity. Unfortunately, it is oriented to words comparison.

The Levenshtein distance (also called edit distance) is a discrete function that, given two strings, measures the minimum number of insertions, deletions and substitutions of single characters needed to transform one string into the other. Its plain, recursive implementation requires $O(3^n)$ time in the size of the strings; however, it is easy to obtain a $O(n^2)$ dynamic programming version.

The idea of extending the Levenshtein distance to compute the similarity between phrases has been explored by some authors. Leusch et al [6] defined a measure, called the inversion edit distance, which is based on measuring clocks reordering (i.e., two sentences are similar if a block of words just changes its position). In 2004, Vilares et al [9] presented an algorithm that combines the edit distance between parse trees and single-term similarity.

2 Similarity Methods

In this section we describe three metrics for measuring phrase similarity. For the sake of space economy we omit the pseudocode of the corresponding algorithms. However, they are available in `ing.utalca.cl/~rangles/mt/pseudocodes.pdf`.

2.1 Basic Metric

Consider the intuitive definition of similarity presented in [8]: "The similarity between two objects A and B is a function of their commonalities and differences". This notion of similarity is formalized in the following definition.

Definition 1 (Simple similarity between phrases). *The similarity between two phrases, f_1 and f_2, is defined as $sim(f_1, f_2) = C - (D + I)$ where C is the number of coincident words, D is the number of words to eliminate, and I is the number of words to insert, all operations required to transform f_1 into f_2.*

This definition yields to a very obvious algorithm to compare two phrases. Despite its simplicity, it reflects well the intuition that two phrases having several words in common (even if the word are permuted) are more similar than those having several different words (i.e. we need to delete or insert words to transform a phrase into the other).

2.2 Word-Oriented Edit Distance

Our second alternative is a rather sophisticated one, inspired on the edit distance. Since a phrase is composed by words, the comparison of two phrases must be based on the words they contain instead of the characters. So, based on the standard version, we define a word-oriented edit distance.

The proposed method considers five types of edit operations: the three basic ones (namely, insertions, deletions or substitutions of *single words*), the transposition of *two adjacent words*, and another operation that allows to *move a word* within a phrase. The latter operation softens the comparison of two phrases that share words in different order. To do so, instead of use the same penalization for every edit operation, we define two values: we penalize with *cIDS* for insertions, deletions and substitutions of single words, and with *cMT* for single word movements and transpositions of two adjacent words. After some preliminary experimental evaluation we set $cIDS = 8$ and $cMT = 2$, as with these values we reflect the intuitive notion of similarity among phrases.

To compute the cost of transforming the phrase $f_1 = w_{11}w_{12}\dots w_{1n}$ into phrase $f_2 = w_{21}w_{22}\dots w_{2m}$, our algorithm takes into account the cost of transforming each of the n prefixes of f_1 into each of the m prefixes of f_2. Using a matrix *cost* of size $(m+1) \times (n+1)$ to manage all the costs of prefix transformation, we can implement an efficient dynamic programming version.

This measure begins by computing size of each phrase $n \leftarrow |f_1|$ and $m \leftarrow |f_2|$. Next, it creates the matrix *cost* of size $m + 1 \times n + 1$. In order to minimize the

cost of transforming the prefix $f_{1,1...j} = w_{11}w_{12}\dots w_{1j}$ into prefix $f_{2,1...i} = w_{21}w_{22}\dots w_{2i}$, we consider several cases: (i) transform prefix $f_{1,1...j}$ into prefix $f_{2,1..i-1}$ and pay the cost of insert word w_{2i}; (ii) transform prefix $f_{1,1...j-1}$ into prefix $f_{2,1..i}$ and pay the cost of delete word w_{1j}; (iii) transform prefix $f_{1,1...j-1}$ into prefix $f_{2,1..i-1}$, and substitute w_{1j} by w_{2i} if they differ or do nothing if they are equal; and (iv) transform prefix $f_{1,1...j-2}$ into prefix $f_{2,1..i-2}$ and pay the cost of swapping words $w_{1(j-1)}$ and w_{1j}. Furthermore, in the third case, if the words differ, but the word w_{2i} belongs to f_1 we account this as a word movement. As we want to minimize, we pick the minimum cost among insertions, deletions, substitutions, movements and word transpositions.

To initialize the matrix *cost* we note that the first row represents the cost of transforming f_1 into the empty phrase, by deleting each of its words. Analogously, the first column represents the cost of transforming the empty phrase in f_2 by inserting its words. The final result is stored in cell $cost_{m,n}$.

2.3 Sequential Distance

One of the drawbacks we detected in the word-oriented edit distance occurs when there are repetitions of words in the target phrase (phrase f_2). So we devise a sequential approach to compare two phrases.

The basic idea is to traverse both phrases simultaneously, word-wise from the beginning and perform edit operations as needed. If the i-th word of both phrases f_1 and f_2 coincide, that is, if $w_{1i} = w_{2i}$, we advance to the next word. Otherwise, we need to determine whether we need to insert, delete or transpose a word. If word w_{1i} does not belong to the suffix of f_2 (which is denoted by $f_{2,i...m}$), we delete w_{1i} from f_1 and pay for the cost of deleting it (cID); else, if w_{1i} does belong, we preserve it to future process. On the other hand, if word w_{2i} belongs to the suffix of f_1 we just fetch it by a circular shifting of the words in the sub-phrase of f_1 limited by i and the position of w_{2i} within the suffix of f_1 (and pay cT for the transposition). Finally, if w_{2i} does not belong, we need to insert it into f_1 (and pay cID for the insertion). Next, we process the next word until consume one or both phrases.

After this processing, it is possible that any of the phrases still has words to process. If so, either all the unprocessed words in f_2 are inserted into f_1 and we pay cID for each insertion; or all the unprocessed words in f_1 are deleted and pay cID for each deletion. Finally, we return the dissimilarity assessment.

After some preliminary experimental evaluation, we set $cID = 8$ and $cT = 2$, as with these values we reflect the intuitive notion of similarity among phrases.

3 Experimental Evaluation

In this section we evaluate the metrics presented above in terms of their effectiveness and efficiency. Recall that effectiveness is purely a measure of the ability of a system to satisfy the user in terms of the relevance of items (words, phrases, documents, etc.) retrieved, whereas efficiency measures the computer resources used by a system to complete the retrieval process.

As usual in the area of Information Retrieval, the effectiveness of a metric is measured in terms of precision and recall. In our context, the *precision* of a metric is the fraction of retrieved phrases that are relevant, while *recall* is the fraction of relevant phrases that are retrieved. Additionally, we consider the notions of coverage and sensitivity. The *coverage* of a metric is the number of results needed to recover all the relevant phrases. The *sensitivity* measures the degradation in the recovering capacity of a metric under specific types of errors (swap, insertion or deletion). Finally, the *efficiency* is the execution time required to rank all the phrases in the corpus according to a given metric. In what follows, we formally define the experiments and discuss the results.

3.1 Experiments

In order to define the experiments we carry out in this paper, consider the following general definitions:

- Assume that C_L denotes a corpus of phrases in a given language L. In this sense, we consider a corpus C_{en} of English phrases (approx. 48.7 MB) and a corpus C_{es} of Spanish phrases (approx. 55.4 MB). Each corpus contains 1 million phrases obtained from DBpedia dumps (`wiki.dbpedia.org/Downloads`) corresponding to Wikipedia articles.
- For each corpus C, we generate a set of 100 test phrases obtained randomly from C (without duplicates).
- Given a test phrase f, we generate 10 phrases from f via different transforming operations, specifically: 1 or 2 swaps, 1 or 2 inserts, 1 or 2 deletes, swap + insert, insert + delete, and delete + swap. Note that, the resulting phrases can be considered relevant as they are obtained from the original test phrase.
- Given two phrases f and f', a metric is a function $M(f, f')$ which returns a value representing the similarity between f and f'. We use M_1 to denote the basic metric (Section 2.1), M_2 for the adapted edit-distance (Section 2.2), and M_3 for the sequential metric (Section 2.3).
- Given a metric M, a corpus C and a test phrase $f \in C$, the function $Rank(M, C, f)$ returns the phrases in C ranked by their similarity with f, i.e. each phrase $f' \in C$ is ranked according to $M(f, f')$. The closer to the top in the rank that a phrase f' occurs, the more similar is f' to the test phrase f.
- Given a sorted list of phrases L, the function $Top_N(L)$ returns the first N phrases in L.

Given a metric M, a corpus of phrases C, a test phrase $f \in C$ and a total number of phrases to retrieve k, we consider the following experiments:

Experiment 1: Calculating Precision. The *precision* of a metric M is given by the fraction (or percentage) of relevant phrases retrieved by M with respect to the total number of phrases retrieved k. Specifically, if s is the number of relevant phrases occurring in $Top_k(Rank(M, C, f))$ then the precision of M is s/k. This

experiment considers to compute, for each metric M_i and corpus C_L, the *average precision* over a set of 100 test phrases obtained from C_L. The precision results are showed in Figures 1(a) and 2(a).

Experiment 2: Calculating Recall. The *recall* of a metric M is given by the fraction (or percentage) of relevant phrases retrieved by M with respect to the total number of relevant phrases. Specifically, if r is the total number of relevant phrases and s is the number of relevant phrases occurring in $Top_k(Rank(M, C, f))$ then the recall of M is s/r. This experiment considers to compute, for each metric M_i and corpus C_L, the *average recall* over the set of 100 test phrases obtained from C_L. Figures 1(b) and 2(b) show the recall results.

Experiment 3: Calculating Coverage. The *coverage* of a metric M is defined as the number (or percentage) of phrases that we need to recover from $Rank(M, C, f)$ in order to obtain all the relevant phrases. The coverage was calculated with a upper bound of 100 recovered phrases. This experiment considers to compute, for each metric M_i and corpus C_L, the *average coverage* over the set of 100 test phrases obtained from C_L.

Experiment 4: Calculating Sensitivity. The *sensitivity* of a metric M under a type of error ξ (swap, insert or delete operation) measures how the efficiency of the metric is affected by the occurrence of multiple errors of type ξ. In this sense, the sensitivity shows the degradation rate of M due to an increasing number of errors of type ξ.

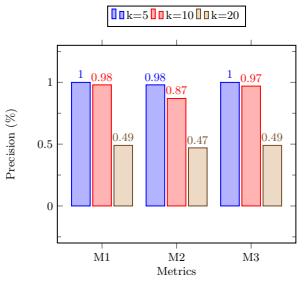
Given a test phrase f and a type of error ξ, we generate "perturbed" phrases from f by introducing multiple errors of type ξ into f, incrementally. Specifically, we construct the set of perturbed phrases $\{f_1, \ldots, f_n\}$ where f_1 contains 1 error, f_2 contains 2 errors, and so on, until completing n errors. The sensitivity of a metric M according to a set of perturbed phrases (as defined above) is calculated as $(Rank(M, C, f_n) - Rank(M, C, f_1))/(n-1)$ where n indicates the maximum number of errors. In our experiments, the sensitivity is calculated with $n = 5$.

This experiment considers to compute, for each metric M_i, corpus C_L and type of error ξ (swap, insert or delete), the *average sensitivity* over the set of 100 test phrases obtained from C_L. The sensitivity results are showed in Figures 1(c) – (e) and 2(c) – (e).

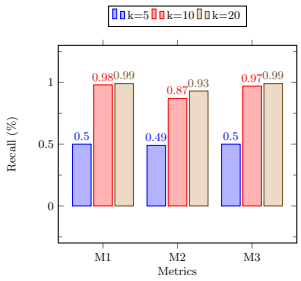
3.2 Results and Comments

Figures 1 and 2 show the results of the experimental evaluation for both English and Spanish, respectively.

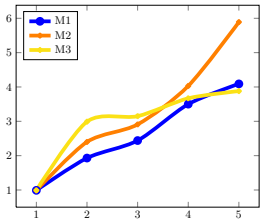
In terms of precision, Figures 1(a) and 2(a) show that all the methods are very similar. Considering that the total number of relevant phrases is 10, the precision decreases when the number of phrases retrieved increases from $k = 10$ to $k = 20$. Specifically, for $k = 5$ the precision is almost 100%, for $k = 10$ the precision is higher that 85%, and for $k = 20$ the precision is mildly lower than 50%. Additionally, we can see that the precisions of M_1 and M_3 are very similar whereas M_2 is slightly lower.



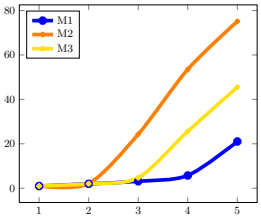
(a) Precision of the metrics.



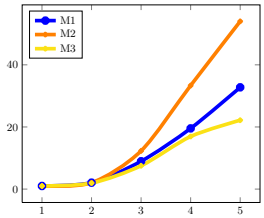
(b) Recall of the metrics.



(c) Sensitivity of the metrics under swaps.

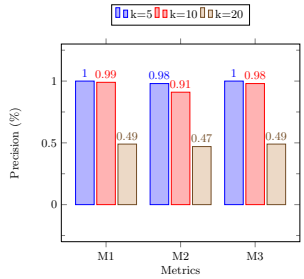


(d) Sensitivity of the metrics under insertions.

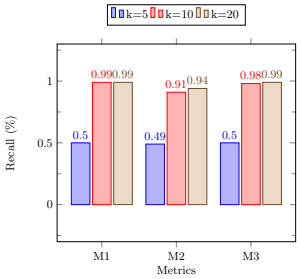


(e) Sensitivity of the metrics under deletions.

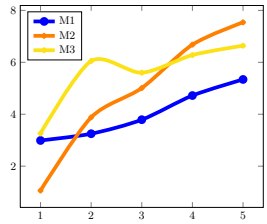
Fig. 1. Experimental evaluation over English corpus. M_1 identifies the basic metric, M_2 the adapted edit-distance and M_3 the sequential metric. The total number of relevant phrases is 20 and k defines the number of phrases retrieved.



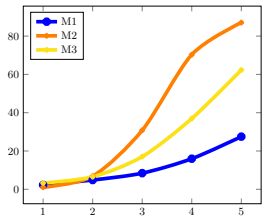
(a) Precision of the metrics.



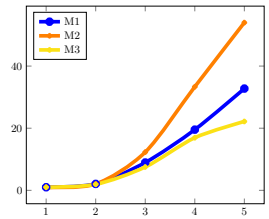
(b) Recall of the metrics.



(c) Sensitivity of the metrics under swaps.



(d) Sensitivity of the metrics under insertions.



(e) Sensitivity of the metrics under deletions.

Fig. 2. Experimental evaluation over Spanish corpus

In Figures 1(b) and 2(b), we can see that the three metrics are very similar in terms of recall. For $k = 5$, all the metrics are able to recover the 50% of the total set of relevant phrases, and for $k >= 10$ the recall is higher than 85%.

With respect to their coverage, we can indicate (the plot is omitted for lack of space) that the three metrics need less or equal than 20 results to recover the set of 10 relevant phrases. Specifically, M_1 requires 11 results, M_2 requires 20 results, and M_3 requires 10 results. In this sense, M_2 requires the double of results to recover all the relevant phrases.

Figures 1(c) – (e) and 2(c) – (e) show different behaviors with respect to the sensitivity to swap, insert and delete errors. M_1 is the least sensitive to swaps and insertions, and M_3 is the least one to deletions. In contrast, M_2 is the most sensitive to the three operations. Note that, the values returned by the sensitivity tests are not included, however the degradation rate is visible in the charts.

Finally, we can indicate that the average execution time (in microseconds) is 14 μs for M_1, 53 μs for M_2 and 9 μs for M_3, considering phrases in English. In the case of Spanish phrases, the times are 23 μs for M_1, 71 μs for M_2, and 19 μs for M_3. Hence, we have that the sequential metric is the most efficient, closely followed by the basic metric, and the adapted edit-distance is the one with the highest average execution time.

4 Conclusions

Finally, we will try to answer the question: Which is the best metric? Considering that there is not a unique and clear definition of similarity between phrases, and that the notion of relevance is very subjective, we conclude that there exist no metric that can be efficient and effective for all the cases, when only considering simple edit word operations. However, from the experiments presented in this paper, we can conclude that a simple metric (as the basic metric defined here) can be more effective and efficient than a complex one (as the word-oriented edit-distance) considering the reduced setup of edit operations.

On the other hand, we hope that by introducing semantic elements in the metrics (such as the use of synonyms or semantic similarity measures between words) or by considering the context of the phrases, we can improve the similarity assessment between the phrases.

References

1. Balasubramanian, N., Allan, J., Croft, W.B.: A comparison of sentence retrieval techniques. In: Proc. 30th ACM SIGIR, pp. 813–814. ACM (2007)
2. Chiang, D.: Hierarchical phrase-based translation. Computational Linguistics 33(2), 201–228 (2007)
3. Hammouda, K.M., Kamel, M.S.: Phrase-based document similarity based on an index graph model. In: Proc. 2002 IEEE ICDM, pp. 203–210 (2002)
4. Hammouda, K.M., Kamel, M.S.: Efficient phrase-based document indexing for web document clustering. IEEE Trans. Knowl. Data Eng. 16(10), 1279–1296 (2004)

5. Koehn, P., Och, F.J., Marcu, D.: Statistical phrase-based translation. In: Proc. 2003 NAACL, vol. 1, pp. 48–54. Assoc. for Computational Linguistics (2003)
6. Leusch, G., Ueffing, N., Ney, H.: A novel string-to-string distance measure with applications to machine translation evaluation. In: MT Summit IX (2003)
7. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10(8), 707–710 (1966)
8. Lin, D.: An information-theoretic definition of similarity. In: Proc. 15th Intl. Conf. on Machine Learning, pp. 296–304. Morgan Kaufmann (1998)
9. Vilares, M., Ribadas, F.J., Vilares, J.: Phrase similarity through the edit distance. In: Galindo, F., Takizawa, M., Traunmüller, R. (eds.) DEXA 2004. LNCS, vol. 3180, pp. 306–317. Springer, Heidelberg (2004)
10. Zens, R., Och, F.J., Ney, H.: Phrase-based statistical machine translation. In: Jarke, M., Koehler, J., Lakemeyer, G. (eds.) KI 2002. LNCS (LNAI), vol. 2479, pp. 18–32. Springer, Heidelberg (2002)