String Distances for Near-duplicate Detection

Iulia Dănăilă, Liviu P. Dinu, Vlad Niculae, and Octavia-Maria Șulea

***Abstract*—Near-duplicate detection is important when dealing with large, noisy databases in data mining tasks. In this paper, we present the results of applying the Rank distance and the Smith-Waterman distance, along with more popular string similarity measures such as the Levenshtein distance, together with a disjoint set data structure, for the problem of near-duplicate detection.**

***Index Terms*—Near-duplicate detection, string similarity measures, database, data mining.**

I. Introduction

A. Motivation

THE concept of *near-duplicates* belongs to the larger class of problems known as *knowledge discovery* and *data mining*, that is identifying consistent patterns in large scale data bases of any nature. Any two chunks of text that have possibly different syntactic structure, but identical or very similar semantics, are said to be near duplicates. During the last decade, largely due to low cost storage capacity, the volume of stored data increased at amassing rates; thus, the size of useful and available datasets for almost any task has become very large, prompting the need of scalable methods. Many datasets are noisy, in the very specific sense of having redundant data in the form of identical or nearly identical entries. In an interview for The Metropolitan Corporate Counsel (see http://www.metrocorpcounsel.com/articles/7757/near-duplicates-elephant-document-review-room), Warwick Sharp, vice-president of Equivio Ltd., a company offering information on retrieval services to law firms with huge legal document databases, noted that 20 to 30 percent of data they work with are actually near-duplicates, and this is after identical duplicate elimination. The most extreme case they handled was made up of 45% near-duplicates. Today it is estimated that around 7% of websites are approximately duplicates of one another, and their number is growing rapidly. On the one hand, near-duplicates have the effect of artificially enlarging the dataset and therefore slowing down any processing; on the other hand, the small variation between them can contain additional information so that, by merging them, we obtain an entry with more information than any of the original near-duplicates on their own. Therefore, the key problems regarding near-duplicates are identification (detection) and aggregation. It is probable that different methods are needed to treat different types of data: for example, small texts, large texts, or images.

Manuscript received on November 15, 2011, accepted for publication on January 6, 2012.

The authors are with the Faculty of Mathematics and Computer Science, University of Bucharest, Romania (e-mail:danailaiulia@yahoo.com, ldinu@fmi.unibuc.ro, vlad@vene.ro, mary.octavia@gmail.com).

Octavia-Maria Șulea is also with the Faculty of Foreign Languages and Literatures, University of Bucharest, Romania.

The work [1] identified the following domains that can benefit from efficient near-duplicate detection and aggregation methods.

- Web mirrors identification
- Clustering for related documents
- Data extraction
- Plagiarism detection
- Spam detection
- Duplicates in domain-specific corpora

These are by no means exhaustive; the problem finds applications in countless fields.

When looking for duplicates in domain-specific corpora, the goal is to identify near-duplicates arising out of revisions, modifications, copying or merger of documents, etc. Example datasets for such an application are TREC benchmarks, Reuters news articles, and Citeseer data (duplicate scientific article citations). See [1, Conrad and Schriber (22)] for a case-study involving legal documents at a law firm. [1, Manber (42)] initiated an investigation into identification of similar files in a file system, with applications in saving disk space. [1, Review (2009)] identifies a few sample situations when we might deem two text documents as being duplicates of each other:

- Files with a few different words - widespread form of near-duplicates
- Files with the same content but different formatting - for instance, the documents might contain the same text, but dissimilar fonts, bold type or italics
- Files with the same content but different file type - for instance, Microsoft Word and PDF versions of the same file.

For short texts such as text messages, [2] indicated the fundamental differences that must be taken into account when doing term weighting, for example. For short messages, larger differences need to be tolerated, and as much semantic information needs to be taken into account. This technique is also relevant for title matching or for comparing short fields from a database. The literature contains various methods, each more suited for specific applications. Depending on the domain and of the specific goals, certain methods are better than others.

A new algorithm could be tailored to a particular task, improving in the measures that have more weight for that particular application, while possibly scoring less from other

points of view: for example, a duplicate detection algorithm for handheld devices is subject to heavy computational and memory limits, so some accuracy needs to be traded. Alternatively, an innovative and general algorithm could improve the state of the art performance in multiple applications, without trading off any resources.

B. State of the art

The state of the art methods in near-duplicate detection cover a broad spectrum of applications and are based sometimes on radically different background techniques. We will first review the web crawling and mining domain and its particular applications. [1] made two research contributions in developing a near-duplicate detection system intended for a multi-billion page repository. Initially, they demonstrated the appropriateness of Charikar's fingerprinting technique [3] for the objective. Locality-sensitive hashing methods have been used in the context of Map-Reduce systems in order to efficiently do approximate nearest neighbour searches in parallel, on big data: this method is taught at Stanford in their class CS246: Mining Massive Data Sets [1]. The major advantage of it is the speed and scalability, while the drawback of this method is the lack of room for tweaking. [4] from Google developed a two-step duplicate identification method that first finds candidates using Charikar's fingerprinting method, followed by refining the query response using similarity measures on the tractable subsets identified by the first step. (US Pat. 8015162). [5] proposed a novel algorithm called I-Match, which they have shown to perform well on multiple datasets, differing in size, document length and degree of duplication. This is step forward, but its drawbacks are that it relies on term frequencies, which can mislead when compared to a ranking-based approach. Secondly, it requires a lexicon, and therefore domain knowledge and language assumptions. For this reason, the system cannot be used out of the box for different problems, but its performance might be better after appropriate tweaking. Another key discussion in duplicate identification is whether to assume the transitivity of the duplicate relation. Granted, this reduces the number of total comparisons needing to be made. Hashing-based detectors use this fact in order to say that objects assigned to the same bucket are duplicates. In practice, however, because we are facing noisy near duplicates, such a procedure can propagate and augment errors.

On the problem of near-duplicate image detection, [6] applied compact data structure to region-based image retrieval using EMD (Earth Mover's Distance) and compared their results positively with previous systems. [7] have applied the neuroscience-inspired Visual Attention Similarity Measure in order to give more weight to regions of interest. A previous, but nonetheless efficient system was given by Chum et al., using locality-sensitive hashing on local descriptors (SIFT), with *tf-idf*-like weighting schemes, which suggests a unified approach based on deep learning, that would work on text and images. An extension of this method is used by Gao and Tang, wherein they initially compare a subset of local features from subsets of two images, followed by crossed near-neighbour searches which should succeed if the images are near-duplicates (US Pat. App 12576236). Furthermore, recent developments in dictionary learning gave way to powerful applications in image classification, denoising, inpainting and object recognition (the Willow team at INRIA [8]). These methods can prove very useful as feature learners for near-duplicate image detection and we intend to leverage them in our system. Andrew Ng and his team at Stanford have successfully applied this kind of unsupervised feature learning and sparse coding, traditionally used in image processing for text processing tasks [9], which encourages the idea that the features for our system can be learned automatically from domain specific data, and thus work efficiently on different types of data.

[1] http://cs246.stanford.edu

C. Our approach

As far as we are aware, there is no research combining deep / unsupervised feature learning with near-duplicate identification and detection. After building a tractable feature-representation of the data, any duplicate detection algorithm needs a notion of similarity. At the moment we sticked with text features, but tried out different metrics. There is a number of metrics used to define similarity [10], around which duplicate detection algorithms are built.

Identification of an adequate metric for determining the similarity of two objects is an intensely studied problem in linguistic and in social sciences. The numerous possible applications (from establishing text paternity, measuring the similarity between languages, text categorization [11]) place this problem in the top of open problems in domains like computational linguistics.

This paper focuses on finding duplicates represented as textual strings. The similarity between two strings is generally measured by Levenshtein (edit) distance or variants. In this paper we use other two distances (Rank distance and Smith-Waterman distance) and compare them. We will introduce them in the following part, along with the union-find disjoint set data structure used to manage the data and optimize the number of comparisons.

Section 3 is dedicated to experimental results, and the final section presents our conclusions and our intended future work.

II. Preliminaries

A. Rank distance

The rank-distance metric was introduced by Dinu in [12] and was successful used in various domains as natural languages similarities, authorship identification, text categorization, bioinformatics, determining user influence [13], etc. To measure rank distance between two strings, we use the following strategy: we scan (from left to right) both

strings and for each letter from the first string we count the number of elements between its position in first string and the position of its first occurrence in the second string. Finally, we sum all these scores and obtain the rank distance. Clearly, the rank distance gives a score zero only to letters which are in the same position in both strings, as Hamming distance does (we recall that Hamming distance is the number of positions where two strings of the same length differ). On the other hand, an important aspect is that the reduced sensitivity of the rank distance w.r. to deletions and insertions is of paramount importance, since it allows us to make use of *ad hoc extensions to arbitrary strings*, such as do not affect its low computational complexity,

When rank distance is restricted to permutations (or full rankings), it is an *ordinal* distance tightly related to the so-called *Spearman's footrule*.

Let us go back to strings. Let us choose a finite alphabet, say $\{A, C, G, T\}$ as relevant for DNA strings, and two strings on that alphabet, which for the moment will be constrained to be a permutation of each other. E.g. take the two strings of length 6, $AACGTT$ and $CTGATA$. To compute rank distance, we proceed as follows: number the occurrences of repeated letters in increasing order to obtain $A_1A_2C_1G_1T_1T_2$ and $C_1T_1G_1A_1T_2A_2$. Now, proceed as follows: in the first sequence A_1 is in position 1, while it is in position 4 in the second sequence, and so the difference is 3; compute the difference in positions for all letters and sum them. In this case the differences are 3, 4, 2, 1, 3, 1 and so the distance is 14. Even if the computation of the rank distance as based directly on its definition may appear to be quadratic, two algorithms which take it back to linear complexity are presented in [14].

Let $u = x_1x_2 \dots x_n$ and $v = y_1y_2 \dots y_m$ be two strings of lengths n and m, respectively. For an element $x_i \in u$ we define its *order* or *rank* by $ord(x_i|u) = i$: we stress that the rank of x_i is its position in the string, counted from the left to the right, *after* indexing, so that for example the second T in the string $CTGATA$ has rank 5.

Note that some (indexed) occurrences appear in both strings, while some other are *unmatched*, i.e. they appear only in one of the two strings. In definition 1 the last two summations refer to these unmatched occurrences. More precisely, the first summation on $x \in u \cap v$ refers to occurrences x which are common to both strings u and v, the second summation on $x \in u \setminus v$ refers to occurrences x which appear in u but not in v, while the third summation on $x \in v \setminus u$ refers to occurrences x which appear in v but not in u.

Definition 1. *The rank distance between two strings u and v is given by:*

$$\Delta(u,v) = \sum_{x \in u \cap v} |ord(x|u) - ord(x|v)| + \sum_{x \in u \setminus v} ord(x|u) + \sum_{x \in v \setminus u} ord(x|v). \quad (1)$$

Example 1. *Let $w_1 = abbab$ and $w_2 = abbbac$ be two strings. Their corresponding indexed strings will be: $\overline{w_1} = a_1b_1b_2a_2b_3$ and $\overline{w_2} = a_1b_1b_2b_3a_2c_1$, respectively. So, $\Delta(w_1, w_2) = \Delta(\overline{w_1}, \overline{w_2}) = 8$*

Remark 1. *The ad hoc nature of the rank distance resides in the last two summations in **(1)**, where one compensates for unmatched letters, i.e. indexed letters which appear only in one of the two strings.*

B. Smith-Waterman Distance

The Smith-Waterman algorithm was introduced in [15], beeing a variation of Needleman-Wunsch algorithm. Since it is a dynamic programming algorithm, it has the desirable property that it is guaranteed to find the optimal local alignment with respect to the scoring system being used (which includes the substitution matrix and the gap-scoring scheme). The main difference to the Needleman-Wunsch algorithm is that negative scoring matrix cells are set to zero, which renders the (thus positively scoring) local alignments visible. Backtracking starts at the highest scoring matrix cell and proceeds until a cell with score zero is encountered, yielding the highest scoring local alignment

For this application, it is not necessary to build string alignement seeing as we are only interested in the final score, so we will exclude this portion for minimizing the execution time. We considered delta = 1 (the cost value for a gap), the matched score = 2 and the unmatched score = -1.

C. Union-Find Algorithm

Under the assumption that the *is-a-duplicate-of* relation is transitive, by building the similarity graph (thresholded according to table I), the problem of near-duplicate detection amounts to finding the connected components of the resulting graph. This way we can avoid unnecessary comparisons between nodes that are already connected, and reduce computations for a memory cost.

The Union-Find structure was proposed for the task of finding and storing connected components in a graph, for the specific task of near-duplicate entry detection, in [16]. This method is based on disjoint sets with a distinguished item in each, called the representative. An implementation of this well-known data structure was used in our experiment.

III. Experimental results

A. Datasets

In this section we will test the near-duplicate text document detection algorithms discussed above on two data bases: one representing a collection of IT products, and the other containing bibliographic entries.

The first database was put together from different online sources [2] to which near duplicates (containing noise in

[2]Data were collected from catalogues such as http://www.cdw.com/, http://www.itproducts.com/ and http://www.streetdirectory.com/.

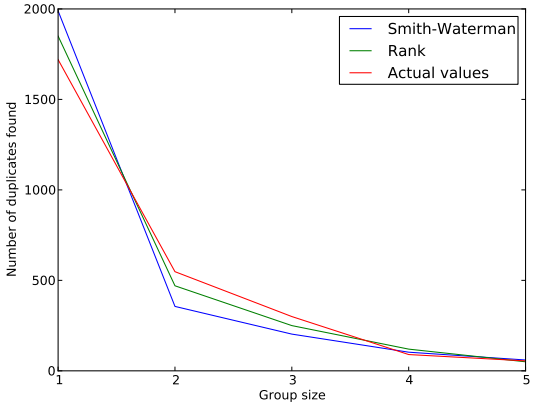

Fig. 1. Results of the first two algorithms on the artificially distorted database, along with the ground truth

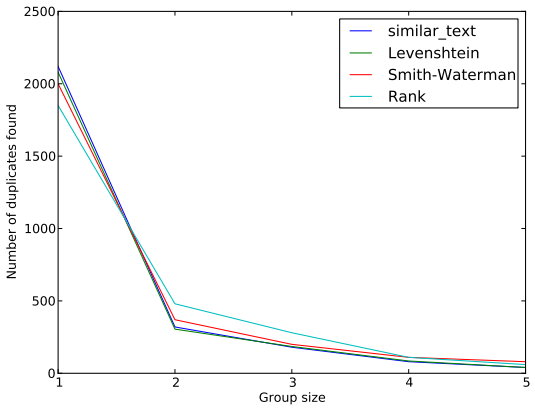

Fig. 2. Results of all similarity algorithms on the artificially distorted database

the form of character insertion, deletion, substitution and transposition) were added.

The second database represents a real-world, undistorted bibliographic collection in BibTeX format, from which we extracted only the title and the author names, in order to lighten the workload given our assumption that the most errors occur in these fields. The source of the data is “A Collection of Computer Science Bibliographies” [17]. Since this collection has over 600,000 entries, we filtered only the ones from the

TABLE I
THE ALGORITHMS USED, AND THE THRESHOLD THAT DEFINES NEAR-MATCHES, WHEN COMPARING STRINGS a AND b, OF LENGTH n_a AND n_b RESPECTIVELY

Metric	Perfect matching	Near matching
Rank distance	$d = 0$	$d \leq \frac{n_a n_b}{2})$
Smith-Waterman	$d = 2\max(n_a, n_b)$	$d \geq \min(n_a, n_b)$
Levenshtein	$d = 0$	$d \leq \frac{\max(n_a, n_b)}{2}$
Similar-text	$p = 100\%$	$p \geq 50\%$

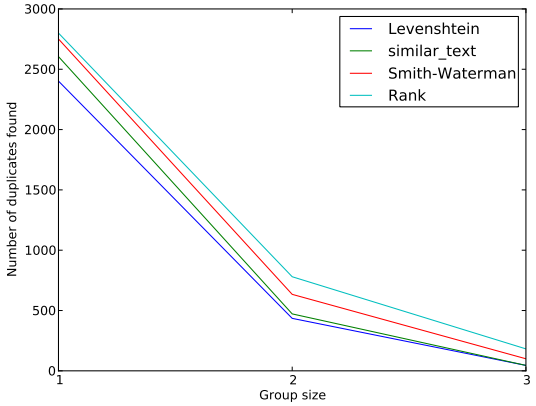

Fig. 3. Results of all similarity algorithms on the bibliography database

TABLE II
DUPLICATES DISTRIBUTION IN ARTIFICIALLY DISTORTED DATA

Group size	Groups number	Input number	Percent
1	1720	1720	62.77%
2	274	548	20.00%
3	96	288	10.51%
4	26	104	3.79%
5	16	80	2.91%
Total	2132	2740	100%

“Planning and Scheduling” category, leaving only 3436 entries such as

{`author: "Andrew G. Barto and S. J. Bradtke and Satinder P.Singh", title: "Learning to Act Using Real Time Dynamic Programming"`}.

A sample duplicate entry of this would be

{`author: "Andrew Barto, J. S. Bradtke and S. P. Singh", title: "Learning to Act Using Realtime Dynamic Programming"`}.

We sought out to investigate the problem of recognizing near duplicates by employing two basic tools: the Union-Find algorithm of grouping data efficiently and the algorithms proposed above. We also looked at the efficiency and the correctness of these algorithms. In what follows we will present the algorithms and the results.

B. Results

The distribution for the duplicates in the artificially distorted database is shown in table II.

The results of the algorithms on the artificial database are displayed in figures 1 and 2 while the results on the real database are shown in figure 3. The figures are distribution plots, the y-value at the position $x = k, k \in \{1, 2, ...\}$ showing the number of documents that can be captured in groups of k. In other words, for $k = 1$ it shows the number of documents

that the algorithm thinks have no duplicates, for $k = 2$ it shows the documents that can be grouped in duplicate pairs, while for $k = 3$ they can be grouped in triples. Note that the points should add up to the total size of the database.

The *similar_text* function used for comparison is the text similarity algorithm from [18], as implemented in the PHP programming language's standard library. It is included as reference because of its accesibility, due to this fact.

In the case of the artificially generated noisy database, we have access to the ground truth. From 1 we can see that the results found by Rank distance are closer to the real distribution of duplicates than the ones found by the Smith-Waterman distance.

For the bibliographic entry database, we assume that the ground truth probability of duplication is lower than in the artificial case. No algorithm found more than 3 duplicate entries for the same information. However under visual inspection, the identified duplicates look correct, confirming the precision of the methods. The Rank distance again seems to have a slower decay rate than the other methods, which can be interpreted as higher recall in the tail of the distribution, assuming a fixed precision.

IV. Conclusions

Our methods for verifying existence of aproximate duplicates exhibit improvement over the previous work in this field. The use of the Union- Find algorithm for grouping the entries significantly reduces the number of comparissons, hightening the efficiency of the general agorithm and its run time. Although it is relyies on the existence of transitivity for the similarity reltion, we have seen that no entries were lost and no errors occured in the grouping of objects.

Until now, the majority of studies on the subject of duplicate detection were based on classic distances, such as Hamming or Levenshtein, yet the results were not always correct. The use of the Smith-Waterman algorithm for strings of characters representing words may seem incertain, taking into consideration that DNA chains are not in the same domain as the one choosen here, yet the results of our experiments show a good performance, an excellent precision and an runtime comparable with classic metrics. Rank distance is usually used for computing distance between ranks, but its adaptation to character strings proved to be fast aqnd precise. We note that there are yet many other metrics and algorithms, which may at first seem unsuitable for a certain problem, but through proper study may prove to be a new solution for a classical problem, possibly even better, faster, and more precise. In our case, the Rank algorithm proved to be more precise than the Smith-Waterman algorithm, being the one closest to the real situation of the duplicates in our datasets.

As future work, we plan to extend these methods in such a way as to minimize the number of comparisons needed, using fingerprinting techniques, as well as to extend them in an unified manner for different data types (images, long text fields, etc.)

V. Acknowledgements

All authors contributed equally to the work presented in this paper. The research of Liviu P. Dinu was supported by the CNCS, IDEI - PCE project 311/2011, "The Structure and Interpretation of the Romanian Nominal Phrase in Discourse Representation Theory: the Determiners."

References

[1] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web crawling," in *Proceedings of the 16th international conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 141–150. [Online]. Available: http://doi.acm.org/10.1145/1242572.1242592

[2] C. Gong, Y. Huang, X. Cheng, and S. Bai, "Detecting near-duplicates in large-scale short text databases." in *PAKDD'08*, 2008, pp. 877–883.

[3] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, ser. STOC '02. New York, NY, USA: ACM, 2002, pp. 380–388. [Online]. Available: http://doi.acm.org/10.1145/509907.509965

[4] M. R. Henzinger, "Finding near-duplicate web pages: a large-scale evaluation of algorithms," in *SIGIR*, 2006, pp. 284–291.

[5] A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe, "Collection statistics for fast duplicate document detection," *ACM Trans. Inf. Syst.*, vol. 20, pp. 171–191, April 2002. [Online]. Available: http://doi.acm.org/10.1145/506309.506311

[6] Q. Lv, M. Charikar, and K. Li, "Image similarity search with compact data structures," in *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, ser. CIKM '04. New York, NY, USA: ACM, 2004, pp. 208–217. [Online]. Available: http://doi.acm.org/10.1145/1031171.1031213

[7] L. Chen and F. Stentiford, "Comparison of near-duplicate image matching," in *Proceedings of the 3rd European Conference on Visual Media Production*, 2006. [Online]. Available: http://discovery.ucl.ac.uk/41711/

[8] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, 2009, pp. 689–696. [Online]. Available: http://doi.acm.org/10.1145/1553374.1553463

[9] A. Maas and A. Ng, "A probabilistic model for semantic word vectors," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[10] M.-M. Deza and E. Deza, *Dictionary of Distances*. Elsevier Science, Oct. 2006. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0444520872

[11] L. P. Dinu and A. Rusu, "Rank distance aggregation as a fixed classifier combining rule for text categorization," in *Proceedings of CICLing*, 2010, pp. 638–647.

[12] L. P. Dinu, "On the classification and aggregation of hierarchies with difererent constitutive elements," *Fundamenta Informaticae*, vol. 55, no. 1, pp. 39–50, 2002.

[13] X. Tang and C. Yang, "Identifing influential users in an online healthcare social network," in *Proc. IEEE Int. Conf. on Intelligence and Security Informatics, 2010 (ISI '10)*, May 2010.

[14] L. P. Dinu and A. Sgarro, "A low-complexity distance for dna strings," *Fundamenta Informaticae*, vol. 73, no. 3, pp. 361–372, 2006.

[15] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195—197, 1981.

[16] A. E. Monge and C. P. Elkan, "Efficient domain-independent detection of approximately duplicate database records," *Engineering*, 1997. [Online]. Available: http://www.cecs.csulb.edu/~monge/research/vldb97.pdf

[17] A.-C. Achilles, "A collection of computer science bibliographies," 1996. [Online]. Available: http://liinwww.ira.uka.de/bibliography/index.html/

[18] I. Oliver, *Programming classics - implementing the world's best algorithms*. Prentice Hall, 1994.