Comparing String Similarity Measures for Reducing Inconsistency in Integrating Data From Different Sources

Sergio Luján-Mora and Manuel Palomar
Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante,
Campus de San Vicente del Raspeig
Ap. Correos 99 – E-03080 Alicante, Spain
{slujan, mpalomar}@dlsi.ua.es

Abstract

The Web has dramatically increased the need for efficient and flexible mechanisms to provide integrated views over multiple heterogeneous information sources. One of the main problems in integrating databases into a common repository is the possible inconsistency of the values stored in them, i.e., the very same term may have different values, due to misspelling, a permuted word order, spelling variants and so on. In this paper, we present an automatic method for reducing inconsistency found in existing databases, and thus, improving data quality. In particular, the objective of our method is integration and standardization of different values that refer to the same term. All the values that refer to a same term are clustered by measuring their degree of similarity. The clustered values can be assigned to a common value that could substitute the original values. The paper describes and compares five different similarity measures for clustering and evaluates their performance on real-world data. The method we propose gives good results with a considerably low error rate and improves our previous works.

Keywords: Data cleaning, Data and information quality, Data migration and integration

1. Introduction

Information fusion is the process of integration and interpretation of data from different sources in order to derive information of a new quality. Integrating databases into a common repository has become a

research topic for many years. Information fusion is a very complex problem, and is relevant in several fields, such as Data Re-engineering, Data Warehouse, Web Information Systems, E-commerce, Scientific Databases, etc.

One of the main problems in integrating databases into a common repository is the possible inconsistency of the values stored in them, i.e., the very same term may have different values. This problem is known as the *field matching problem*, and it is part of the Data Cleaning task. This is an important issue, because erroneous datasets propagate error in each successive generation of data.

The problem of the inconsistency found in the values stored in databases may have two principal causes:

1. Databases may contain duplicate values concerning the same real-world entity (inconsistency) because of data entry errors (misspelling, typing errors), because of unstandardized abbreviations, or because different people can use different values to name the same term (spelling variants, permuted word order, transliteration differences). For instance, a database that stores the names of the departments of a university may have several different forms (e.g., the use of upper-case letters or abbreviations): "*Department of Software and Computing Systems*", "*Dep. of Software and Computing Systems*", "*D. of software and computing systems*", etc.
2. On the other hand, this is a common problem in environments where multiple databases must be combined: equivalent data in the multiple sources must be identified (knowledge discovery, data mining, data warehouse, data re-engineering, etc.).

In Figure 1, we present an example to show the aim of our proposal. Let us suppose that we have different databases (particularly, different relational tables) and the sources have different criteria for representing values in affiliation names. For example, with reference to the affiliation of researchers who work at the University of Alicante, we may easily find that there are different values for this university: *"Universidad de Alicante"* or *"Universidad Alicante"* (in Spanish) and *"Alicante University"* (in English).

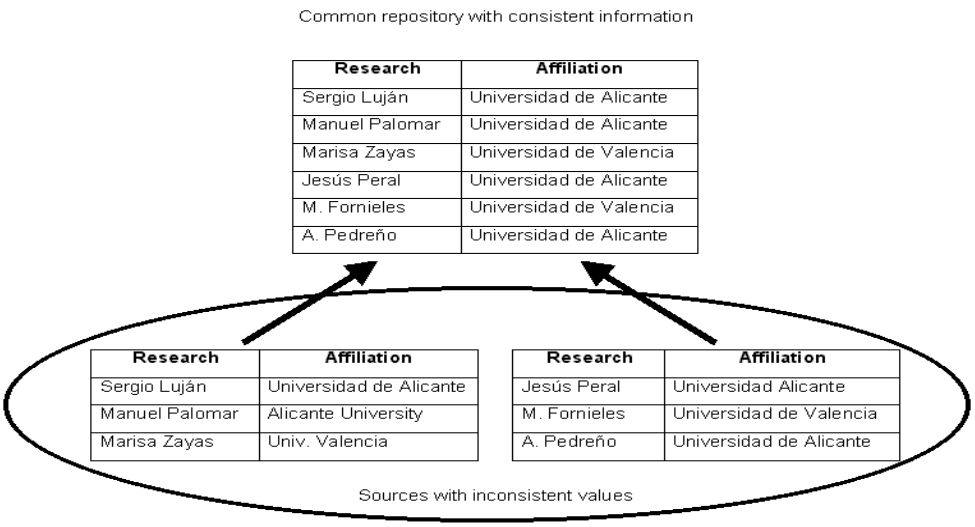

Figure 1. Solving inconsistency into a common repository

The remainder of the paper is structured as follows: Section 2 outlines the origin of the problem and the possible causes that give rise to the different variants that appear for the same term; Section 3 introduces our method for reducing inconsistency found in existing databases; Section 4 explains the core of our study and details the technical aspects of our method; Section 5 provides an evaluation of the method; and finally, our conclusions are presented in Section 6.

2. Analysis of the Problem

After analysing several databases with information both in Spanish and in English, we have noticed that the different values that appear for a given term are due to a combination of the following causes:

1. The omission or inclusion of the written accent: "*Asociación Astronómica*" or "*Asociacion Astronomica*".
2. The use of upper-case and lower-case letters: "*Department of Software and Computing Systems*" or "*Department of software and computing systems*".

3. The use of abbreviations and acronyms: "*Department of Software, University of Alicante*" or "*Dep. of Software, Univ. Alicante*".
4. Word order: "*Miguel de Cervantes Saavedra*" or "*Cervantes Saavedra, Miguel de*".
5. Different transliterations[1]: "*Kolmogorov*" or "*Kolmogorof*", "*Chebyshev*" or "*Tchebysheff*".
6. Punctuation marks (e.g., hyphens, commas, semicolons, brackets, exclamation marks, etc.): "*Laboratorio Multimedia (mmlab)*" or "*Laboratorio Multimedia – mmlab*".
7. Errors: Misspelling (apart from the written accent), typing or printing errors (absence of a character, interchange of adjacent characters, etc.): "*Gabinete de imagen*" or "*Gavinete de imagen*", "*Bill Clinton*" or "*Bill Klinton*".
8. Numbers: "*Area 51*" or "*Area fifty-one*".
9. Extra words: "*Royal Yacht Club*" or "*Yacht Club*".
10. Different denominations: "*Seismological Register Unit*" or "*Seismic Register Unit*".
11. Use of different languages: "*Tribunal de Cuentas*" (Spanish), "*Court of Auditors*" (English) or "*La Cour des Comptes*" (French).

There has been great interest in studying the quality of the information stored in databases for a long time [12, 13, 14], and diverse methods have been developed for the reduction of the inconsistency found in databases [1, 2, 3, 5, 6, 10, 11].

3. Intuitive Proposal of a Method to Reduce the Inconsistency Found in Databases

[1] Transliteration is the general process of converting characters from one particular script to another one, such as converting from Greek to Latin, or Japanese katakana to Latin. It is very important to note that this transliteration is not translation. It is simply converting the letters from one script to another, not translating the underlying words.

The method we propose in this paper improves our previous works [10]. We have added new distances, developed different evaluation measures and employed a different clustering algorithm. These improvements result in a better performance of the method.

Our algorithm resolves all the problems detailed in Section 2, except the three last causes, which depend on how different the two strings that represent the same term are. The method that we propose can be divided into six steps:

1. *Preparation*. It may be necessary to prepare the strings before applying the clustering algorithm.
2. *Reading*. The following process is repeated for each of the strings contained in the input file:

   Read a string

   Expand abbreviations and acronyms[2]

   Remove accents: e.g., *A* substitutes *Á* and *À*, and *a* substitutes *á* and *à*

   Shift string to lower-case

   Store the string: If it has been stored previously, its frequency of appearance is increased by one unit
3. *Sorting*. The strings are sorted, in descending order, by frequency of appearance.
4. *Clustering*. The most frequent string is chosen and it is compared to the rest of the strings, using a measure of similarity. This process is repeated, successively, until all the strings have been clustered.
5. *Checking*. The resulting clusters are verified and the possible errors are located and corrected.
6. *Updating*. The original database is updated. The strings of a cluster are replaced by its *centroid* (the centroid is representative of all the strings in its cluster).

4. Technical Description of the Method

[2] It is in general impossible to expand all the abbreviations: often names are represented by initials, sometimes by only some of the initials, etc.

In this section, technical aspects of our method are described. We start by introducing a previous processing for obtaining better results in Section 4.1. Section 4.2 describes how the similarity between two strings is considered. Section 4.3 presents the algorithm itself and finally, Section 4.4 explains the last step of the method, i.e., checking that the obtained clusters are correct.

4.1. Previous Processing

The strings undergo a previous processing to obtain better results from the clustering. The objective of this processing is to avoid the three first causes of the appearance of different forms for the same term (see Section 2.1.): i.e., accents, lower-case/upper-case and abbreviations. The accents are eliminated, the string is converted to lower-case and the abbreviations are expanded.

4.2. String Similarity

The similarity between any two strings must be evaluated. There are several similarity measures; in our research, we employ five measures: Levenshtein distance (LD), invariant distance from word position (IDWP), a modified version of the previous distance (MIDWP), Jaccard's coefficient (JC), and the minimum of the four previous measures (CSM).

The *edit distance* or *Levenshtein distance* (LD) [8] has been traditionally used in approximate-string searching and spelling-error detection and correction. The LD of strings x and y is defined as the minimal number of simple editing operations that are required to transform x into y. The simple editing operations considered are: the insertion of a character, the deletion of a character, and the substitution of one character with another. In our method, we have taken a unitary cost function for all the operations and for all of the characters. The LD of two strings m and n in length, respectively, can be calculated by a dynamic programming algorithm [7]. The algorithm requires $\Theta(mn)$ time and space.

If two strings contain the same words (variant forms of the same term) but with a permuted word order, the LD will not permit their clustering. To solve this problem, we introduce another distance that we call the *invariant distance from word position* (IDWP) [9]. It is based on the *approximate word matching* referred to in [2, 3]. To calculate the IDWP of two strings, they are broken up into words (we consider a word to be any succession of digits and letters of the Spanish alphabet). The idea is to pair off the words so that the *sum* of the LD is minimised. If the strings contain different numbers of words, the cost of each word in excess is the length of the word.

We also use a *modified IDWP* (MIDWP). We add a new matching condition: if two strings fulfil Equation 1, we assume they match perfectly (in that case, we consider their LD is zero).

$$LD(x, y) \leq 1 + \frac{|x| + |y|}{20}. \quad \textbf{(1)}$$

The last similarity measure we have employed is the *Jaccard's coefficient* (JC) [15], the ratio of the matching words in x and y to all the words in x and y:

$$JC = \frac{|X \cap Y|}{|X \cup Y|}, \quad \textbf{(2)}$$

where X is the set of words of the string x and Y the set of words of y.

In order to compare the above-mentioned measures, we need the JC subtracted from one (1 – JC). Besides, the LD, IDWP, and MIDWP are divided by the length of the longest string. Thus, all the measures obtain a similarity value from 0 (x and y are the same string) to 1 (x and y are totally different).

Finally, we also combine the four previous similarity measures (*combined similarity measure*, CSM): we choose the minimum of the four similarity measures for every pair of strings.

4.3. Algorithm

The goal of clustering is to find similarity between strings and cluster them together based on a threshold of similarity between the strings.

In previous works [2, 3, 10], the clustering algorithm employed is basically the *leader algorithm* [4]. This algorithm is chosen as opposed to more elaborate algorithms (e.g. *k-means algorithm*, *Fisher algorithm*) because they are slower and the number of clusters is unknown. The *leader algorithm* is very fast, requiring only one pass through the data, but it has several negative properties: the partition is not invariant under reordering of the cases, the first clusters are always larger than the later ones and the final number of clusters depends on the threshold values. This is due to the very algorithm: the comparison between a new string and the existing clusters is made only until a cluster that meets the condition is found, without considering the possibility that a better value of the criteria is met later, for another cluster.

The clustering algorithm we propose in Table 1 resolves the previous problem: it uses a *centroid* method and the comparison for every string is made with all the existing clusters for the time being.

The algorithm chooses the strings, from greater to smaller frequency of appearance, since it assumes that the most frequent strings have a greater probability of being correct, and thus, they are taken as being representative of the rest. As seen in Table 1, it depends on one parameter α (threshold). The algorithm makes one pass through the strings, assigning each string to the cluster whose *centroid* is closer and close enough (distance between the string and the *centroid* lower than α) and making a new cluster for cases that are not close enough to any existing *centroid*. The distance D is calculated using one of the similarity measures explained in Section 4.2.

The *centroid* of a cluster must be recalculated every time a new string is assigned to the cluster. The *centroid* is chosen to minimise the sum-of-squares criterion:

$$\sum_{i=1}^{n} (D(s_i, C))^2 , \qquad \textbf{(3)}$$

where *n* is the number of strings assigned to the cluster and *C* is the *centroid* of the cluster.

Table 1. Clustering algorithm

Input:
S: Sorted strings in descending order by frequency ($\mathbf{s_1 \ldots s_m}$)
α: Threshold

Output:
C: Set of clusters ($\mathbf{c_1 \ldots c_n}$)

Variables:
b, d, i, j, k, l

STEP 1. Begin with string $\mathbf{s_i}$ (**i** = 1). Let the number of clusters be **k** = 1, classify $\mathbf{s_i}$ into the first cluster $\mathbf{c_k}$.

STEP 2. Increase **i** by 1. If **i** > **m**, stop.

STEP 3. Begin working with the cluster $\mathbf{c_j}$ (**j** = 1). Calculate the distance between the string $\mathbf{s_i}$ and the centroid of cluster $\mathbf{c_j}$: **d** = D($\mathbf{s_i}$, $\mathbf{c_j}$). Let the best cluster be $\mathbf{c_b}$ (**b** = 1).

STEP 4. Increase **j** by 1. If **j** > **k**, then go to *Step 7*.

STEP 5. If D($\mathbf{s_i}$, $\mathbf{c_j}$) < **d**, then let the lower distance be **d** = D($\mathbf{s_i}$, $\mathbf{c_j}$) and the best cluster be **b** = **j**.

STEP 6. Return to *Step 4*.

STEP 7. If **d** < α, assign string $\mathbf{s_i}$ to cluster $\mathbf{c_b}$; recalculate the centroid of cluster $\mathbf{c_b}$ and return to *Step 2*.

STEP 8. Increase **k** by 1. Create a new cluster $\mathbf{c_k}$ and classify $\mathbf{s_i}$ into the new cluster. Return to *Step 2*.

4.4. Revision and Updating

The final step of the method consists of checking the obtained clusters and detecting possible errors to correct them. In the original database, the strings of a cluster are replaced by its *centroid* (it represents its cluster). Therefore, all variants of a term are put together under a single form. Thus, in searching processes, final users will be confident that they have located all values relating to the required term.

5. Experimental Results and Evaluation

We have used four files for evaluating our method[3]. They contain data from four different databases with inconsistency problems: files A, B, and D contain information in Spanish, while file C in English. The method has been implemented in *C* and *C++*, running in *Linux*.

5.1. File Descriptions

Table 2 gives a description of these four files. The *optimal number of clusters* (ONC) indicates the number of handcrafted clusters. The three last columns contain the number of single strings (not duplicated) with (W) and without (WO) the expansion of abbreviations, and the rate of reduction (on expanding the abbreviations, the number of single strings is reduced, since duplicates are removed). We have done all the tests with and without the expansion of abbreviations.

Table 2. File descriptions

File	Size (Bytes)	ONC	Strings in file	Strings WO	Strings W	Reduction (%)
A	10,399	92	234	234	145	38.0
B	1,717,706	92	37,599	1,212	1,117	7.8
C	108,608	57	2,206	119	118	0.8
D	24,364	226	596	584	505	13.5

The amount of duplication of a file is measured by the *duplication factor* (DF) [1], which indicates how many duplicates of each record appear in the file, on the average. Table 3 shows the DF of the four files. It also shows the reduction of the DF when the abbreviations are expanded.

[3] These test files are available at `http://www.dlsi.ua.es/~slujan/files/clusterfiles.tgz`.

Table 3. Duplication factor

File	DF WO	Standard deviation	DF W	Standard deviation	Reduction (%)
A	2.54	1.00	1.58	0.84	37.8
B	13.17	5.39	12.14	5.23	7.8
C	2.09	2.88	2.07	2.80	0.9
D	2.58	1.16	2.23	1.08	13.6

The DF shows how many duplicates of each record exist, but it does not indicates if the duplicates look like each other or not. We have developed a coefficient (consistency index) that permits the evaluation of the complexity of a cluster: the greater the value of the coefficient is, the more different the strings that form the cluster are. A null value indicates that the cluster contains only one string. The *consistency index* (CI) of a cluster of *n* strings is defined as:

$$CI = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} LD(x_i, x_j)}{\sum_{i=1}^{n} |x_i|}. \quad \textbf{(4)}$$

The *file consistency index* (FCI) of a file that contains *m* clusters is defined as the average of the consistency indexes of all the existing clusters in the file:

$$FCI = \frac{\sum_{i=1}^{m} CI_i}{m}. \quad \textbf{(5)}$$

The FCI of the files A, B, C, and D are shown in Table 4. As the FCI is an average, the table also shows the standard deviation. It is obvious that the clusters of file B are more complex than those of files A , C, and D. In all cases, however, the FCI is reduced when expanding the abbreviations, since the discrepancies between the strings of a given cluster tend to diminish. With respect to file C, the reduction of FCI when the abbreviations are expanded is minimum, because the reduction of strings is not appreciable: only 0.8% versus 38.0% (file A), 7.8% (file B), and 13.5% (file D) as it is shown in Table 2.

Table 4. File consistency indexes

File	FCI WO	Standard deviation	FCI W	Standard deviation	Reduction (%)
A	0.31	0.29	0.12	0.26	61.2
B	1.72	1.26	1.11	1.14	35.4
C	0.33	1.18	0.31	1.13	6.0
D	0.53	0.44	0.27	0.27	49.0

5.2. Evaluation Measures

We have evaluated the quality of the produced clusters when our method is applied by using four measures that are obtained by comparing the clusters produced by our method with the optimal clusters (ONC):

1. NC: number of clusters. Clusters that have been generated.
2. NCC: number of completely correct clusters. Clusters that coincide with the optimal ones: they contain the same strings. From this measure, we obtain *Precision*: NCC divided by ONC.
3. NIC: number of incorrect clusters. Clusters that contain an erroneous string. From this measure, we obtain the *Error*: NIC divided by ONC.
4. NES: number of erroneous strings. Strings incorrectly clustered.

NC and NCC versus Threshold for File A with (W) and without (WO) expansion of abbreviations, using the CSM, are plotted in Figure 2. The ONC is the horizontal line. The expansion of abbreviations diminishes NC and increases NCC.

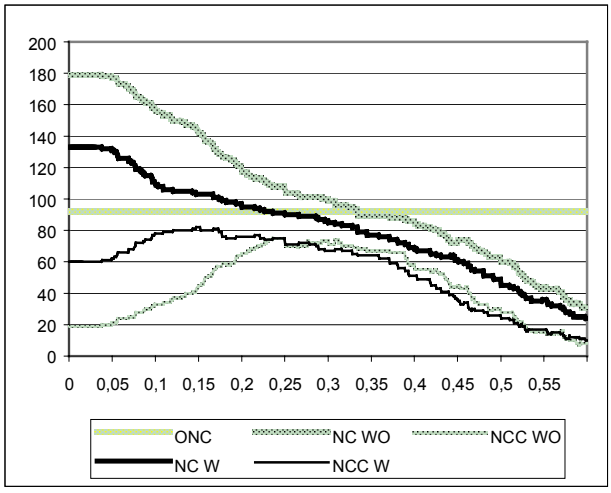

Figure 2. NC and NCC vs. Threshold. File A with and without expansion of abbreviations (CSM)

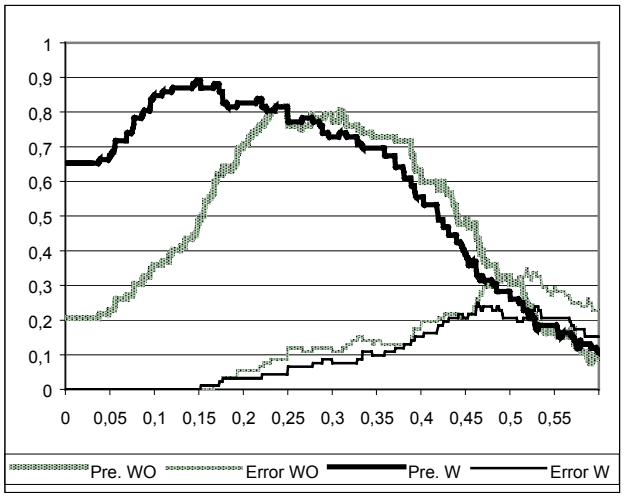

Figure 3. Precision and Error vs. Threshold. File A with and without expansion of abbreviations (CSM)

5.3. Evaluation and Discussion

As we have already mentioned, the clustering algorithm depends on one parameter (α). We have done all the tests on setting its value from 0.0 to 0.599, in 0.001 steps.

We compare the performance of the five similarity measures. The result of the experiments using the four files are shown in Tables 5, 6, 7, 8, and 9. The tables show the highest precision rate and the corresponding error obtained in each file when the LD, IDWP, MIDWP, JC, and CSM are used. The corresponding threshold (α) also appears.

Note that the expansion of abbreviations improves the precision and diminishes the error. Moreover, the best precision, with a lower error, is obtained at a lower threshold when the abbreviations are expanded.

Table 5. LD

File		α	Precision (%)	Error (%)
A	WO	0.311	76.0	8.6
	W	[0.146, 0.151]	83.6	0
B	WO	[0.272, 0.281]	68.4	9.7
	W	[0.276, 0.281]	73.9	5.4
C	WO	[0.159, 0.199]	84.2	1.7
	W	[0.100, 0.127]	84.2	0
D	WO	[0.429, 0.433]	49.5	19.9
	W	[0.320, 0.321]	63.2	11.9

Table 6. IDWP

File		α	Precision (%)	Error (%)
A	WO	[0.334, 0.344]	81.5	10.8
	W	[0.160, 0.166]	84.7	0
B	WO	[0.347, 0.369]	64.1	11.9
	W	[0.341, 0.342]	67.3	8.6
C	WO	[0.143, 0.227]	82.4	1.7
	W	[0.072, 0.119]	82.4	0
D	WO	[0.435, 0.437]	53.0	15.9
	W	[0.218, 0.222]	72.1	3.0

As you can see in Table 7, file A obtains the higher precision (89.1%) when the MIDWP with the expansion of abbreviations is employed. File B obtains the higher precision (77.1%) with the CSM and the expansion of abbreviations (Table 9). As seen in Table 8, file C obtains it (89.4%) when the JC without the expansion of abbreviations is used. Finally, file D achieves the best precision (76.1%) when the MIDWP with the expansion of abbreviations is used (Table 7).

Table 9 shows highest precision and the corresponding error obtained for the four files when the CSM is employed. Files A and C have better precision than file B because their clusters are less complex: files A and C have a FCI around 0.3, whereas file B has a FCI of 1.7 (WO) and 1.1 (W). However, the file D obtains a smaller precision than the file B, although its FCI is smaller.

Table 7. MIDWP

File		α	Precision (%)	Error (%)
A	WO	[0.276, 0.277]	80.4	9.7
	W	[0.153, 0.166]	89.1	0
B	WO	[0.381, 0.382]	66,3	16.3
	W	[0.369, 0.370]	68,4	8.6
C	WO	[0.143, 0.227]	82.4	1.7
	W	[0.072, 0.119]	82.4	0
D	WO	0.388	52.2	16.3
	W	0.187	76.1	2.6

Table 8. JC

File		α	Precision (%)	Error (%)
A	WO	[0.400, 0.416]	72.8	6.5
	W	[0.286, 0.299]	85.8	0
B	WO	[0.500, 0.538]	56.5	17.3
	W	[0.455, 0.461]	58.6	7.6
C	WO	[0.471, 0.499]	89.4	1.7
	W	[0.471, 0.499]	87.7	1.7
D	WO	[0.500, 0.533]	52.6	13.2
	W	[0.385, 0.399]	73.4	3.5

Table 9. CSM

File		α	Precision (%)	Error (%)
A	WO	[0.236, 0.249]	81.5	8.6
	W	[0.147, 0.151]	89.1	0
B	WO	[0.270, 0.288]	71.7	9.7
	W	[0.174, 0.176]	77.1	2.1
C	WO	[0.143, 0.199]	84.2	1.7
	W	[0.097, 0.119]	84.2	0
D	WO	[0.385, 0.387]	52.6	17.2
	W	[0.195, 0.199]	75.2	3.9

In Table 10, we show the precision and error obtained in our previous works [10] and in the current work (the best precision for each file). The test files A, B, C, and D are the same of this paper. As you can see, the new method achieves better results: the precision increases and the error keeps very similar values or even diminish.

Table 10. Precision and Error in previous works

File		Previous works		Current work	
		Precision (%)	Error (%)	Precision (%)	Error (%)
A	WO	70.7	7.6	81.5	8.6
	W	84.8	0	89.1	0
B	WO	67.4	8.7	71.7	9.7
	W	72.8	6.5	77.1	2.1
C	WO	85.9	1.7	89.4	1.7
	W	84.2	1.7	87.7	1.7
D	WO	43.8	16.8	53.0	15.9
	W	67.7	6.2	76.1	2.6

We compare the effect of the expansion of abbreviations in Figure 3. It shows Precision and Error versus Threshold for File A with (W) and without (WO) expansion of abbreviations using the CSM. It is seen that the expansion of abbreviations produces the maximum precision (90%) at a threshold of 0.15. From a threshold of 0.25, the expansion of abbreviations does not influence the precision as observed in the figure. Also, note that there is not error when the threshold is lower than 0.15.

Figure 4 shows Precision versus Threshold for File C without expansion of abbreviations using different similarity measures. The JC obtains the maximum value (90%). All the measures, except the JC, have a similar behaviour: they start at the same level (75%), rise until 85% and then plunge until 20%. However, the JC remains steady over 75% for all the threshold values.

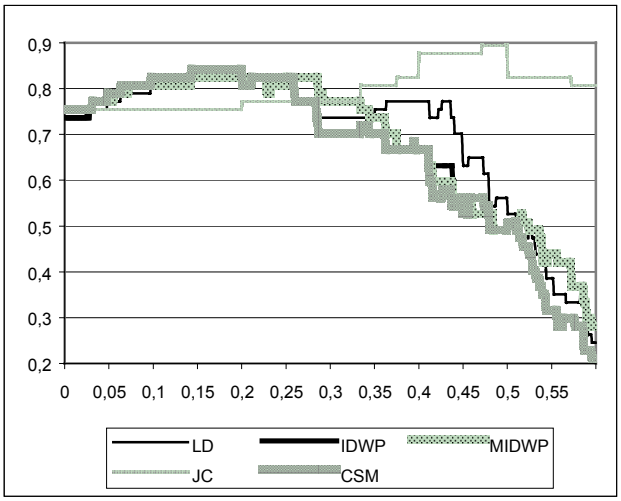

Figure 4. Precision vs. Threshold. File C without expansion of abbreviations (different measures)

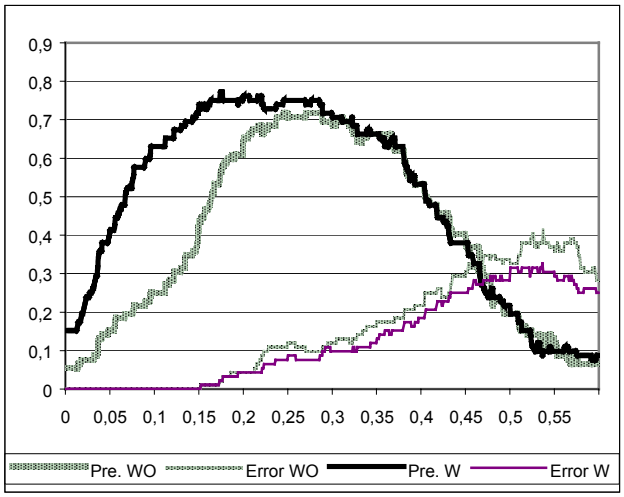

Figure 5. Precision and Error vs. Threshold. File B with and without expansion of abbreviations (CSM)

Finally, from Figure 5 it can be again seen that the expansion of abbreviations influences the precision at a low threshold, but from a threshold of 0.25, the influence is imperceptible. Also, note that there is not error when the threshold is lower than 0.15. The behaviour is very similar to Figure 3.

6. Conclusions and Work in Progress

Referential integrity provided by relational database management systems prevents users or applications from entering inconsistent data. Databases with an inadequate design may suffer data redundancy and inconsistency. This paper has discussed techniques for improving data quality by clustering different values that refer to the same term and replacing them with a unique form. So, we have presented an automatic method for reducing on the inconsistency found in existing databases. The method we have proposed achieves successful results with a considerably low error rate, although it does not eliminate the need to review the clusters obtained.

The expansion of abbreviations improves on the results in most cases, but we have detected some cases in which it actually makes the results worse. In addition, we have seen that the combined use of four similarity measures (*Levenshtein distance*, *invariant distance from word position*, *modified IDWP*, and *Jaccard's coefficient*) normally obtains the best performance.

The final number of clusters and the effectiveness of the method strongly depends on the threshold value fixed by the user. A very small threshold (conservative) will produce a large number of small clusters and a decrease in the number of matching values that should be clustered, meanwhile a very large (aggressive) one will produce a small number of large clusters and an increase in the number of falsely matched values. Based on the data obtained in our research, we propose the use of a threshold between 0.1 and 0.25.

Our first contribution is an algorithm that is domain-independent and language-independent. Previous related work deals with special cases of the field matching problem (customer addresses, census records, bibliographic databases, etc.). The second contribution is the use of two methods for evaluating the similarity between two strings: the invariant distance from word position, derived from the Levenshtein distance, and the combined similarity measure. Last but not least, we present the consistency index that permits the evaluation of the complexity of a cluster: the greater the value of the coefficient is, the more different the strings that form the cluster are.

Currently, we are working on improving the algorithm in order to cluster the multilingual values. We are applying dictionaries and other techniques relating to natural language processing (e.g., removing stop words, lexical analysis).

References

1. Dina Bitton, David J. DeWitt. Duplicate record elimination in large data files. *ACM Transactions on Database Systems*, 8(2):255-265, 1983.

2. James C. French, Allison L. Powell, Eric Schulman. Applications of Approximate Word Matching in Information Retrieval. In Forouzan Golshani, Kia Makki, editors, *Proceedings of the Sixth International Conference on Information and Knowledge Management (CIKM 1997)*, pages 9-15, Las Vegas (USA), November 1997.

3. James C. French, Allison L. Powell, Eric Schulman, John L. Pfaltz. Automating the Construction of Authority Files in Digital Libraries: A Case Study. In Carol Peters, Costantino Thanos, editors, *Proceedings of the First European Conference on Research and Advanced Technology for Digital Libraries (ECDL 1997)*, Lecture Notes in Computer Science, vol. 1324, pages 55-71, Springer-Verlag, 1997.

4. John A. Hartigan. *Clustering Algorithms*. A Wiley Publication in Applied Statistics. John Wiley & Sons, New York (USA), 1975.

5. Mauricio Antonio Hernández, Salvatore J. Stolfo. The Merge/Purge Problem for Large Databases. In Michael J. Carey, Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 127-138, San Jose (USA), June 1995.

6. Mauricio Antonio Hernández, Salvatore J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Journal of Data Mining and Knowledge Discovery*, 2(1):9-37, 1998.

7. Daniel S. Hirschberg. Serial Computations of Levenshtein Distances. In Alberto Apostolico, Zvi Galil, editors, *Pattern Matching Algorithms*. Oxford University Press, New York (USA), 1997.

8. Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10:707-710, 1966.

9. Sergio Luján-Mora. An Algorithm for Computing the Invariant Distance from Word Position. Available at `http://www.dlsi.ua.es/~slujan/files/idwp.ps`, June 2000.

10. Sergio Luján-Mora, Manuel Palomar. Clustering of Similar Values, in Spanish, for the Improvement of Search Systems. In Maria Carolina Monard, Jaime Simão Sichman, editors, *IBERAMIA-SBIA 2000 Open Discussion Track Proceedings*, pages 217-226, Sao Paulo (Brazil), November 2000.

11. Alvaro E. Monge, Charles P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'97)*, pages 23-29, May 1997.

12. Amihai Motro, Igor Rakov. Estimating the Quality of Databases. In T. Andreasen, H. Chistiansen, H.L. Larsen, editors, *Proceedings of FQAS 98: Third International Conference on Flexible Query Answering Systems*, Lecture Notes in Artificial Intelligence, vol. 1495, pages 298-307, Springer-Verlag, 1998.

13. Edward T. O'Neill, Diane Vizine-Goetz. Quality Control in Online Databases. *Annual Review of Information Science and Technology*, 23:125-156, 1988.

14. Edward T. O'Neill, Diane Vizine-Goetz. The Impact of Spelling Errors on Databases and Indexes. In Carol Nixon, Lauree Padgett, editors, *National Online Meeting Proceedings*, pages 313-320, New York (USA), May 1989.

15. Cornelis Joost van Rijsbergen. *Information Retrieval*. Butterworhs, second edition, London (UK), 1979.