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

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. 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, in principle, could substitute the original values. We evaluate different similarity measures for clustering. The method we propose gives good results with a considerably low error rate.

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. The problem of inconsistency has also lately been a focus of interest in the area of Data warehouses (DW) as a DW is a repository of integrated information from distributed, autonomous, and possibly heterogeneous, sources.

Traditional search systems work by matching the term that is being searched with the values stored in the corresponding database. If the information contained in databases is inconsistent (i.e., if a given term appears with different values because several denominations exist, or because it is misspelled), a search using a given value will not provide all the available information about the term.

In Figure 0, 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).

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

1. If the number of possible values that a single field can accept is not controlled, a given person, (or different persons), may insert the same term with different values. 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): "*Departamento de Lenguajes y Sistemas Informáticos*", "*Depto. de Lenguajes y Sistemas Informáticos*", "*Dpt. de lenguajes y sistemas informáticos*", etc.
2. When we try to integrate different databases into a common repository (e.g., in a DW), one or more of them may suffer from the above-mentioned problem. The consistency of their contents has been guaranteed separately. However, the criteria used for establishing the consistency of each one might well be different and integrating them all could cause inconsistency problems. For example, we wish to integrate three different databases that store bibliographical information. The authors might well appear in different forms in each one: i.e., full names, "*Miguel de Cervantes Saavedra*", or by last names first and then the first name, "*Cervantes Saavedra, Miguel de*", or by first name and last name only, "*Miguel de Cervantes*".
3. Another problem is the multilinguality. In a multilingual society (e.g., European Community) it is common to find official names written in different languages. For instance, we consult a database that stores information about university researchers, (e.g., researcher's name, researcher's university, etc.), and we wish to obtain a list of all of the researchers who work at the University of Alicante. We may easily find that there

are different values for this university: "*Universidad de Alicante*" (in Spanish), "*Universitat d'Alacant*" (in Catalan), "*University of Alicante*" or "*Alicante University*" (in English), and "*Université d'Alicante*" (in French).

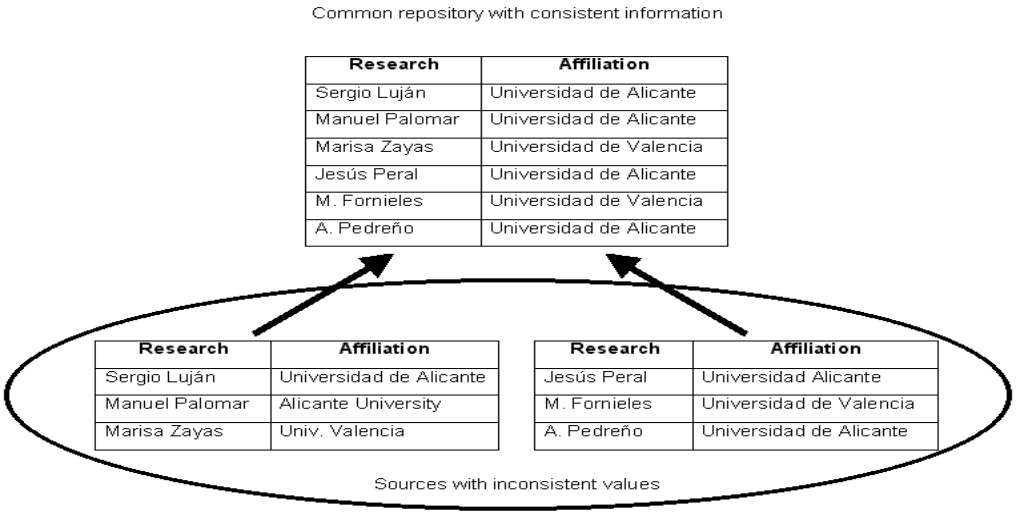

Figure 0. Solving inconsistency into a common repository

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: "*Departamento de Lenguajes y Sistemas Informáticos*" or "*Departamento de lenguajes y sistemas informáticos*".
3. The use of abbreviations and acronyms: "*Dpto. de Derecho Civil*" or "*Departamento de Derecho Civil*".
4. Word order: "*Miguel de Cervantes Saavedra*" or "*Cervantes Saavedra, Miguel de*".
5. Different denominations: "*Unidad de Registro Sismológico*" or "*Unidad de Registro Sísmico*".
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*".
8. Use of different languages: "*Universidad de Alicante*" (Spanish) or "*Universitat d'Alacant*" (Catalan).

There has been great interest in studying the quality of the information stored in databases for a long time [8, 9, 13], and diverse methods have been developed for the reduction of the inconsistency found in databases [11, 12].

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

The method we propose in this paper improves our previous works [7] that were developed from French's automatic creation of authority files for bibliographical catalogues [1, 2]. 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 fifth and the eighth, 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[1]

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*.

4. Technical Description of the Method

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).

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

The *edit distance* or *Levenshtein distance* (LD) [5] has been traditionally used in approximate-string searching and spelling-error detection and correction (causes 6 and 7). 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 [4]. 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 (cause 4), 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) [6]. It is based on the *approximate word matching* referred to in [1]. 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).

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

The last similarity measure we have employed is the *Jaccard's coefficient* (JC) [10], 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 [1, 2, 7], the clustering algorithm employed is basically the *leader algorithm* [3]. 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.

Table 1. Clustering algorithm

Input:
S: Sorted strings in descending order by frequency ($\mathbf{s_1 \ldots s_m}$)
$\boldsymbol{\alpha}$: 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** < $\boldsymbol{\alpha}$, 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*.

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.

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 three files for evaluating our method. They contain data from three different databases with inconsistency problems: files A and B 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 three 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 and without 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 (W) and without (WO) 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

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:

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

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:

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

The FCI of the files A, B and C are shown in Table 3. 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 file A and C. 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) and 7.8% (file B), as it is shown in Table 2.

Table 3. File consistency indexes

File	FCI WO	Standard deviation	FCI W	Standard deviation
A	0.311	0.298	0.127	0.269
B	1.726	1.267	1.113	1.142
C	0.337	1.181	0.319	1.136

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:

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.

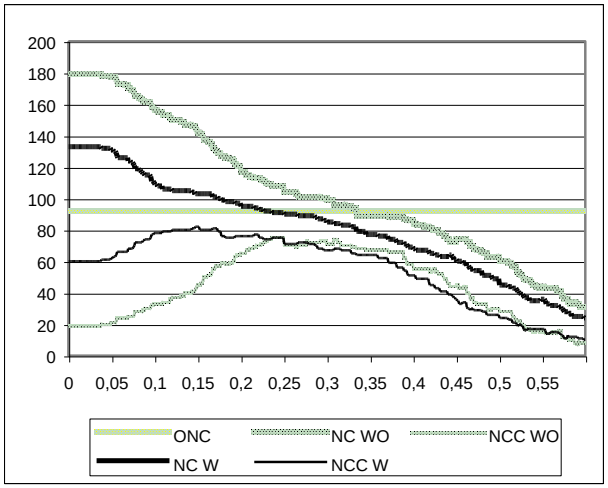

Figure 1. NC and NCC vs. Threshold. File A with and without expansion of abbreviations (CSM)

NC and NCC versus Threshold for File A with (W) and without (WO) expansion of abbreviations, using the CSM, are plotted in Figure 1. The expansion of abbreviations diminishes NC and increases NCC.

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 files A and C are shown in Tables 4, 5, 6 and 7. The tables show the highest precision rate and the corresponding error obtained in each file when the LD, IDWP, MIDWP and JC 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.

Table 4. LD

File		α	Precision (%)	Error (%)
A	WO	0.311	76.0	8.6
	W	[0.146, 0.151]	83.6	0
C	WO	[0.159, 0.199]	84.2	1.7
	W	[0.100, 0.127]	84.2	0

Table 5. IDWP

File		α	Precision (%)	Error (%)
A	WO	[0.334, 0.344]	81.5	10.8
	W	[0.160, 0.166]	84.7	0
C	WO	[0.143, 0.227]	82.4	1.7
	W	[0.072, 0.119]	82.4	0

As you can see in Table 6, File A obtains the higher precision (89.1%) when the MIDWP with the expansion of abbreviations is employed. However, as seen in Table 7, File C obtains it (89.4%) when the JC without the expansion of abbreviations is used.

Table 6. MIDWP

File		α	Precision (%)	Error (%)
A	WO	[0.276, 0.277]	80.4	9.7
	W	[0.153, 0.166]	89.1	0
C	WO	[0.143, 0.227]	82.4	1.7
	W	[0.072, 0.119]	82.4	0

Table 7. JC

File		α	Precision (%)	Error (%)
A	WO	[0.400, 0.416]	72.8	6.5
	W	[0.286, 0.299]	85.8	0
C	WO	[0.471, 0.499]	89.4	1.7
	W	[0.471, 0.499]	87.7	1.7

Table 8 shows highest precision and the corresponding error obtained for files A, B, and C 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).

Table 8. 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

In Table 9, we show the precision and error obtained in our previous works [7]. The test files A, B and C are the same of this paper. If this table is compared to Table 8, you can see the new method achieves better results: the precision increases and the error keeps very similar values or even diminish.

Table 9. Precision and Error in previous works

File		Precision (%)	Error (%)
A	WO	70.7	7.6
	W	84.8	0
B	WO	67.4	8.7
	W	72.8	6.5
C	WO	85.9	1.7
	W	84.2	1.7

We compare the effect of the expansion of abbreviations in Figure 2. It shows Precision 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.

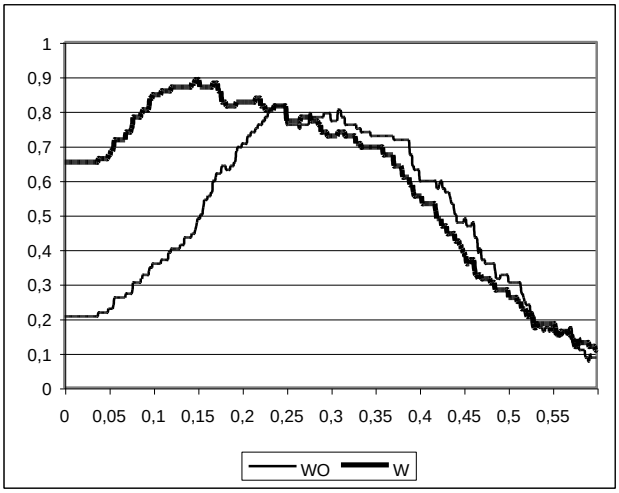

Figure 2. Precision vs. Threshold. File A with and without expansion of abbreviations (CSM)

Figure 3 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.

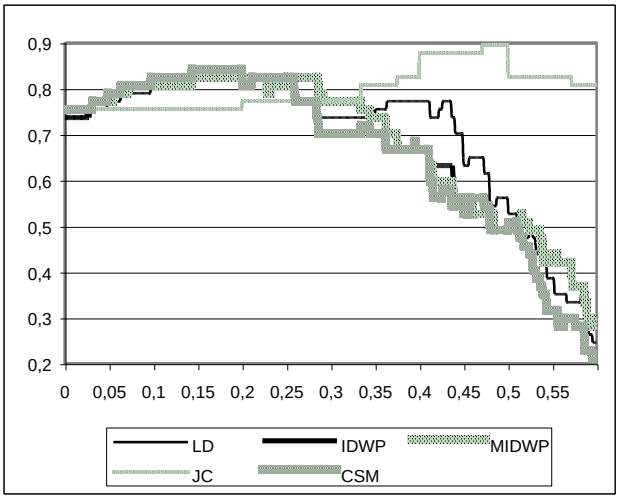

Figure 3. Precision vs. Threshold. File C without expansion of abbreviations (different measures)

Finally, from Figure 4 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 (the behaviour is very similar to Figure 2). Also, note that there is not error when the threshold is lower than 0.15.

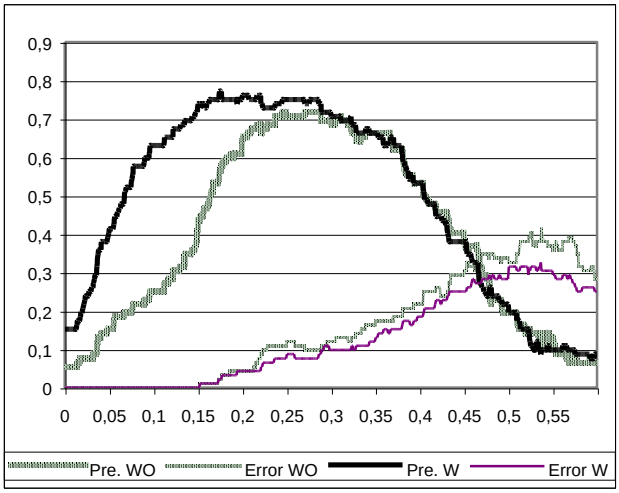

Figure 4. Precision and Error vs. Threshold. File B with and without expansion of abbreviations (CSM)

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 no 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 strongly depends on the threshold value fixed by the user. A very small threshold (conservative) will produce a large number of small clusters, meanwhile a very large (aggressive) one will produce a small number of large clusters. Based on the data obtained in our research, we propose the use of a threshold between 0.1 and 0.25.

Other algorithms like *k-means* can not be applied to this problem because the number of clusters is unknown (k-means requires the number of clusters to be specified beforehand).

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. 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.

2. 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)*, pages 55-71, Pisa (Italy), September 1997.

3. John A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, New York (USA), 1975.

4. D.S. Hirschberg. Serial Computations of Levenshtein Distances. In A. Apostolico, Z. Galil, editors, *Pattern Matching Algorithms*. Oxford University Press, 1997.

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

6. 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.

7. 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, appear in *IBERAMIA-SBIA 2000 Open Discussion Track Proceedings*, pages 217-226, Sao Paulo (Brazil), November 2000.

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

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

10. C.J. van Rijsbergen. *Information Retrieval*. Butterworhs, London (UK), 1979.

11. 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.

12. 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, 1997.

13. 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, Springer-Verlag, 1998.