

LSI-03-9-R

Searching by Approximate Personal-Name Matching

Rafael Camps y Jordi Daudé
Departamento LSI
Universitat Politècnica de Catalunya
Barcelona

1.- Introduction

- Object
- Errors
- Graphic and phonetic similarity
- First name and surnames
- Similarity functions and search techniques
- Historical view and state-of-the-art
- Contributions

2.- Designing a distance function: DEA

- Distance functions
- Metric distances
- Discrimination
- Positions
- Thresholds
- Other distance functions
- Posiciones de las operaciones

3.- Evaluating and comparing distance functions

- Objective evaluation
- The *MiFa* graphic
- Data Volumes, β -Factor and Precision
- Example
- Two surnames
- The *Recall/Precision* graphic
- Efficacy: *E* and *J*
- Summary of the comparison between distance functions

4.- Searching technique

- Object
- Basic algorithm for edit distances with costs
- Improving the basic algorithm: cut-off column
- Improving the search: trie-tree
- Evaluation of the efficiency

5.- Conclusions

6 - References

Searching by approximate personal-name matching

Rafael Camps and Jordi Daudé (April 2003)

*Software Department, Universitat Politècnica de Catalunya, C/ Jordi Girona 1-3, Barcelona, Spain
08034. E-mail: rcamps@lsi.upc.edu*

SUMMARY

We discuss the design, building and evaluation of a method to access the information of a person, using his name as a search key, even under the presence of errors and noises. We present a similarity function, the DEA function, based on the probabilities of the edit operations accordingly to the involved letters and their position, and using a variable threshold. The efficacy of DEA is quantitatively evaluated, without human relevance judgments, very superior to the efficacy of known methods. A very efficient approximate search technique for the DEA function is also presented based on a compacted trie-tree structure.

KEY WORDS: approximate name searching; approximate string matching; edit distance; trie-tree

1. INTRODUCTION

Object

Data related to people is stored in almost all Information Systems (IS): customers, patients, taxpayers, drivers, authors, etc. Expansion of *Internet* contributes to increase the data about people into IS. Very often there is the need to find the data of a person in a database (DB) using his/her name as a search key. But we can have difficulties in finding the person because the name may contain errors. For example, if we look in a DB for the information of a person that we think his/her surname is *Blasco*, it may occur that in the DB that person appears as *VELASCO*. Therefore, we need that the system recognizes that both strings are probably variants or errors, one of each other. We also need that this approximate searching of names in the DB, is made in a sufficiently short time.

We use the abbreviation APNM (*Approximate Personal-Name Matching*) for the location of people by their name, in structured DB, tolerating the presence of errors. The aim of this work is to present appropriate methods for APNM.

Errors

The experience says that in DB of IS, that are usually structured and high volume, the percentage of people with errors in their names is rarely below 3% and it is not unusual to reach levels close to 30%. It is usually considered that the IS of the Spanish Public Administration, have in their DB around a 20% of personal records with errors in names. There are similar figures for the North American Administration [1]. If we suppose that names are introduced in the system via a keyword and the set of first name and surnames has an average of about 20 characters, then a 20% of people with some kind of error mean a typing error ratio of 1% approximately, which is within the usual limits of quality in non-verified data. According to Barker [2] 50% of personal-names introduced

by Internet users, contain errors. The problem is very common in all type of applications, and in fields as diverse as Health Services, Marketing by mail, Customers Relationship Management, Justice, Treasury, Information Retrieval for libraries, Police, Census agencies, etc.

Frequently and specially in the Public Administration, the sources of data and their ways to reach the IS, can be very diverse and therefore the causes of errors or noise can also be numerous. These anomalies are usually grouped in two families, *phonetics* and *graphics*. That is, related to sounds (errors because of deficiencies of speech or hearing, ignorance of the language, etc) and related to graphics (errors in typing, in OCR devices, in the visual interpretation of manual writing, etc). The same anomaly can be produced by different causes. For example; the confusion between letters *M* and *N* can be phonetic or a confusion because its manual writing is similar or a typing error since they are neighboring in qwerty keyboards.

The people involved in the use of a IS, have a good knowledge of the domain of values for most of the attributes of the DB: cities of the delegations, names of products, salaries, etc. In these cases usually they make only typing errors. But when the domain of values is not known, as is the case for the surnames, many errors of all types are made. Ideas of interest on the causes of errors and noises can be found in [3] [4] [5] and [6].

Graphic and phonetic similarity

There are two traditional approaches for the criterion or function of similarity between two strings of characters:

- *Graphical similarity* (or physical similarity). The similarity is determined by the character-wise comparison of both strings, processing its coincidences or differences and calculating a distance.
- *Phonetic Similarity*. The words are seen as sequences of sounds. Usually the similarity is determined by means of some phonetic codification system (as the popular *Soundex* system [7]) whose objective is that the same code corresponds to two words, if, and only if, they are phonetically similar words. These systems usually consist of a set of rewriting rules. The phonetic approach has been (and still is) the most used for the case of personal-names [8] [9] [10] [5] [11] [12].

We have added phonetic characteristics to a criterion of comparison of characters, as Veronis [3] and Zobel [13] also did. The set of the multiple causes of errors and its interaction, does not allow us to describe the problem in a deterministic way. This lead us to probabilistic approaches based on experimental data obtained from real corpora.

First name and surnames

In Spain, complete names are formed (according to the law) by three parts; first name and two surnames. There are important differences between the lexical characteristics of first names and those of surnames. The number of different surnames in a DB can be very high, but the number of first names is rather limited. First names usually are much more known than surnames for the people involved in the IS and for that reason less phonetic or interpretation errors are made in them. On the contrary, first names have variants and they are abbreviated with much greater frequency than surnames.

For all this, in the APNM systems, first names are submitted to a different treatment than surnames. Usually a dictionary is built with the acceptable first names, and all the variants and usual errors known until that moment, with all needed cross-references. But this is not possible in the case of surnames, being in them where the greater difficulty of the APNM resides. This work is centered in solving the problems with surnames because they are more frequent and

more difficult to solve. Anomalies of the macrostructure, such as transposition between parts of a personal-name, are not contemplated in this work, but they are well studied in [14][15][16]. From now on, we will use the term *name* and *surname* indistinctly. We will use the term *first name* only when really needed.

Similarity functions and search techniques

There is no doubt that between the names *VELASCO* and *BLASCO* there is more proximity or similarity than between *VELASCO* and *MARTIN*. In this work we will say that two names are similar, if with a certain probability both refer to the same person. In the pair *VELASCO* / *BLASCO* very probably one of the two strings is consequence of anomalies in the oral transmission or the writing of the other. But it is not probable that this is the case of the pair *VELASCO* / *MARTIN*. Note that we want to find the person that we are looking for, if it exists, independently of which of the two names (the one stored in the DB or the one in the query) or no one, is correct.

In practice and from a functional point of view, the APNM can appear to us with varied aspects. The variant that we will adopt in this work can be expressed as follows: Given a name x and a set of names C_1 , obtain a set $C_2 \subseteq C_1$ with those names that are similar to x . The names from C_1 as well as x , are not necessarily correct. The C_2 set may be empty. We can introduce a parameter k to tune the similarity criterion. This variant of the problem can be expressed as a function:

$$C_2 = \text{Similar} (x, C_1, k)$$

This function is based on the calculation of a measurement of dissimilarity or *distance* $\delta(x, y)$. In approximate searching we tend to imagine the greater or smaller proximity between two words, like a minor or greater distance in a certain space (using these terms in an informal way). In this work, we are interested on a distance that relates to the probability that x does not refer to the same person than y .

The C_2 set could be obtained ordered by ascendant values of the distance, and k would be the acceptable maximum value, the threshold, of that distance. The C_1 set usually is a vocabulary of names existing in the DB in which we want to search. Then, the C_2 subset obtained with the function *Similar*, will be used to access to the rest of information of the persons that have exactly the name x or a similar one.

In APNM we can differentiate two basic aspects; one of logical level, *What criterion do we adopt for the similarity?* and another one of more physical level, *How do we implement the search?*

What? We want to find a criterion or function that determines if two names are similar or that shows us a quantification of its similarity. That determination has to be effective, that is, as much correct as possible.

How? We want to find a technique that allows us to locate in a DB, in a sufficiently fast way, the information of all the people whose names are similar (accordingly to the similarity function) to the searched name. We want to find an efficient implementation of the *Similar* function. We need a suitable data structure, a search strategy in that structure and an efficient algorithm for the determination of similarity. To simplify we will call this *search technique*.

To find a valid function, that is, "sufficiently effective", is very difficult. The problem is in finding a distance function $\delta(x,y)$ that captures the actual name anomalies. The methods proposed until now are far from achieving it. The discrepancies between the computable distances and the "reality", produces misidentification (false negatives) and overidentification (false positives). Distance functions are usually more effective than phonetic codification methods. Nevertheless, distance

functions have the disadvantage that the associated search techniques are usually very inefficient, very time consuming. Unless for a small DB, the sequential search, analyzing all the names of the DB, is excessively expensive. Therefore, our basic goal is two fold: to find an *effective* similarity function, and an *efficient* search technique.

Historical view and state-of-the-art

Before 1980 some works of interest were published about the problem of the approximate searching of personal-names in the DB of IS, the APNM problem. We stand out Davidson [17] Taft [8] and Fokker [18]. But from 1980, symptoms of frustration appeared. As Hernansen [19] said in his PhD thesis, the problem is "*exceedingly difficult*", and there seems to be no way to solve it in a general way.

In 1980 an interesting survey was published on the very general subject of the Approximate String Matching (ASM), written by Hall & Dowling [20]. The APNM is discussed only slightly as a special case within the broad world of ASM. In 1992 another survey was published with some interest for the APNM field, written by Karen Kukich [21] it shows a complete "state-of-the-art" of ASM, automatic correction and related subjects.

In the 90's, the term ASM, that until then was being used in its wider sense, begins to be used in a very restricted sense, limiting it to the study of efficient algorithms related to edit distances. The efficacy is not considered because in most applications (mainly in Biology) the approach is deterministic.

Recently, in 2001 a *survey* from Gonzalo Navarro [22] on the ASM subject has appeared, basically focused in the simple distance (the simple distance is the minimum number of edit operations necessary to transform a string into another).

We will now mention the works published from the early 80's, that dedicate special attention to the APNM problem.

- *Getty's Synoname and its cousins: A survey of applications of personal name-matching algorithms*, from C.L. Borgman and S.L. Siegfried [5]. It deals with the state-of-the-art of APNM. It emphasizes the systems in real production and explains the multicultural problems of APNM that appear in an archive of History of Art (Getty Foundation). See also [24] and [25].

- *Searching proper names in databases* [12] and *Retrieval effectiveness of proper name search methods* [26] both of U.Pfeifer. It is an experimental, and subjective, comparison between several methods for determination of the similarity of personal-names.

- *Phonetic string matching: Lessons from Information Retrieval* from J.Zobel and P.Dart [13]. It compares many criteria of similarity applicable to words, but its authors question the results; one of its conclusions is that the traditional method for efficacy evaluation (use of human judges) is not appropriate.

- *Similarity Searching in the CORDIS Text Database* from E.G.M. Petrakis and K. Tzeras [27]. It compares several distance functions, for the access to the CORDIS DB of the European Union using personal-names. Like other works, it uses subjective criteria for the evaluation of efficacy (a human judge).

- *Matchsimile: A Flexible Approximate matching Tool for Personal Names Searching* de G.Navarro et al [28]. This recent work, describes a commercial tool for the names searching.

An interesting field, useful to APNM, is the one of names classification according to its ethnic-linguistic origin [29]. In the IS of very multicultural contexts, with great diversity of origins, it can be useful to have diverse criteria of similarity and to have a classification step to direct the process towards the suitable criterion.

Contributions

Perhaps, the two more important drawbacks in the APNM area are:

- The efficacy of the proposed similarity criteria is not high enough for the needs of most applications. And the search techniques with highest efficacy are not efficient because they use to go exhaustively through all the names in the DB. On the other hand, the major efficiency is obtained with the criteria that has the lower efficacy (phonetic codification)
- The evaluations of the efficacy are subjective (*relevance judgments*) [10] [5] [13] [26] [27]

In the next section we present the DEA similarity function. It is an edit distance function but with costs based on the probability of each operation, depending on the involved letters and their position. The distance threshold is not a fixed value but it varies with the length of the searched name. Its efficacy, objectively evaluated, is very high; for example, a *recall* of 94% produces a *fallout* of only 0.2% (section 4).

In section 4 we present an efficient similarity search technique, for the DEA distance function based on a compact trie-tree.

2. DESIGNING A DISTANCE FUNCTION: DEA

Distance functions

The most popular distance (or dissimilarity measure) between two character strings, is defined as the minimum number of edit operations, insert \mathbb{I} , delete \mathbb{D} and substitution \mathbb{S} , needed to transform one string into the other. This distance is named simple edit distance or *simple distance* for short [22]. Transforming *GIMENEZ* into *JIMNEEZ* can not be done with less than 3 edit operations; for example, a \mathbb{S} (G to J), a \mathbb{D} (of an E) and an \mathbb{I} (of another E). So, the simple distance between these two names is:

$$\delta(GIMENEZ, JIMNEEZ) = 3$$

Note that the above transformation sequence is not the only possible one with three edit operations, for example the following sequence is also valid; a \mathbb{S} (G to J), another \mathbb{S} (E to N) and a third \mathbb{S} (N to E , but in another position).

It is usually required that the sequence has no more than one operation in the same position. Otherwise, the distance would be not always computable [22] [30].

The simple distance accounts for the physical aspects of the string, so it looks more graphic than phonetic. But a function based on the simple distance can solve some phonetic problems, as for example; it can accept the omission of the sound of a final S (a \mathbb{D} operation), the transformation of the sound of a B into a P (an \mathbb{S} operation), the transformation of *DE LA HOZ* into *DELOZ* (this pair needs a threshold = 4 to be considered similar), etc.

In [31] the transposition operation, T, was introduced, and it is often used in ASM. Additional operation types were proposed in [32] [33] [34] [35].

Here we propose a distance function: DEA. It is an edit distance for which we define a variable threshold depending on the length of the searched name, and with variable costs according to a probabilistic model that tries to catch the errors that actually occur in a corpus, whatever the causes are. The operation costs depend on:

- the type of operation (I, D or S)
- the position where the operation is applied
- the letters involved in the operation

The calculation of a distance usually assumes a previous transformation of the characters in order to obtain a normalized string format that depends on the application. We apply to the names a normalization process that consists basically in:

- a) turn lowercases to uppercases,
- b) diacritics deletion,
- c) compact the contiguous blanks to a single one,
- d) deletion of other symbols than letters or blanks.

Metric distances

Although distance functions have better efficacy than the phonetic codification methods, the later are usually used in big volume DB because they allow the use of efficient techniques, as for example B-trees or hashing. With the distance functions, a sequential total search is usually applied, which is too much time consuming. More efficient algorithms require a metric search space, so the distance must be a metric distance.

Let be Σ the alphabet (the set of accepted characters). Then Σ^* is the set of all possible names, including the void name ϵ . A distance function is a metric in Σ^* , if the following proprieties apply :

$$\left. \begin{array}{l} \delta(x,x) = 0 \\ 0 < \delta(x,y) \quad \text{if } x \neq y \\ \delta(x,y) = \delta(y,x) \\ \delta(x,y) + \delta(y,z) \geq \delta(x,z) \end{array} \right\} \quad \forall x,y,z \in \Sigma^*$$

Costs

Wagner [36] defined a distance function where the three classical edit operations (I, D, S) can have different costs depending on the characters. For example, a substitution of an M by an N can have an assigned cost lower than a substitution of M by R . If the costs of the edit operations are all the same, the distance is metric. The cost of each operation in the simple distance is equal to 1, so the simple distance is always metric.

Some authors have proposed limited costs for specific applications [31] [35]. When the costs of the operations are not limited, it is called a *generalized edit distance* [37].

The value of an edit distance $\delta(x,y)$, can be defined as the minimum cost of all the possible sequences that transform x into y . The cost of a sequence of operations is the sum of their costs. The costs of operations are non-negative real numbers, that we will note as δ_c . If each character is seen as a string of length=1, the distance between two of these strings is $\delta_c(x,y)$. If we include the void string, ϵ , then:

$$\begin{aligned}
\forall x, y \in \Sigma \\
I_x \text{ cost is } \delta_c(\epsilon, x) \\
D_x \text{ cost is } \delta_c(x, \epsilon) \\
S_{xy} \text{ cost is } \delta_c(x, y) \text{ for } x \neq y
\end{aligned}$$

In almost every proposed distance, the insertion of a character into a string is functionally equivalent to a deletion of that character from the other string. And the substitution of x by y , is equivalent to the substitution of y by x . Therefore:

$$\begin{aligned}
\delta_c(x, \epsilon) &= \delta_c(\epsilon, x) \\
\delta_c(x, y) &= \delta_c(y, x)
\end{aligned}$$

How can we determine the cost that we have to assign to each elementary operation? The diversity of proposed answers is a sign of the difficulty of the question [38] [39] [40] [41] [33]. Some researchers try to solve the problem with automatic optimization techniques. For example, some of them apply automatic learning techniques on training corpora, using neural networks [42].

There is no doubt that the costs should depend in some way on the characteristics of the errors (graphic, phonetic, etc) that the system must accept. As far as any type of errors can occur and we do not have a total knowledge about the world to be modeled (the world of errors and variants) we use a probabilistic model from experimental data. We refer to the obtained costs, as DEA costs.

Discrimination

Most of published works about approximate searching that use weighted distances, propose costs inversely proportional to the probability of the operations (or to its logarithm [33]). However, this approach does not take into account the prior probability. The fact that in a corpus of errors there are more substitutions of A by E than substitutions of D by T , should be balanced by the fact that in personal-names the letter A appears much more often than the letter D .

Our approach to estimate the costs is based on the discrimination concept. Let us call *pairs-with-error* a set of pairs of similar names (one pair member is an erroneous version of the other) and *pairs-without-error* a set of pairs of independent names (each one refers to a different person). We call *discrimination* of an edit operation, the ratio between the probability of its occurrence in the set of *pairs-without-error* and the probability of occurrence in the set of *pairs-with-error*. Note that this idea is the same that under the name of *discrimination power* is used in the automatic classification field, where the target is to maximize the ratio between interclass differences and intraclass differences.

We will use as *pairs-with-error* corpus, a *TEST* file containing 10593 pairs of surnames, in such way that one surname is an error or variant of the other. The pairs are real cases obtained from a mailing company. As *pairs-without-error* corpus, we will use a *CONTROL* file containing 9345 pairs, obtained randomly pairing surnames of a list resulting from a mix of the left hand and right hand surnames of the *TEST* pairs (without eliminating duplicates) but deleting from *CONTROL* the pairs already existing in *TEST* or having $\delta = 0$.

The discrimination D_{op} of an edit operation (op) is given by:

$$D_{op} = \frac{Pr(op \text{ in } CONTROL)}{Pr(op \text{ in } TEST)}$$

The numerator of the discrimination $D_s(xy)$ for the substitution operation, can be approximated by the probability that a randomly chosen pair from *CONTROL*, contains a substitution of x by y . This probability can be estimated by:

$$P(xy) = P(x) P(y) \quad \forall x, y \quad x \neq y$$

being $P(x)$ and $P(y)$ the relative frequencies of x and y , respectively, in the *CONTROL* file. Note that $P(xy) = P(yx)$.

The denominator of the discrimination $D_s(xy)$, is the probability $P_s(xy)$ of a substitution of x by y obtained from the pairs in *TEST*. Therefore, we can obtain the discrimination $D_s(xy)$ of a substitution, as:

$$D_s(xy) = \frac{P(xy)}{P_s(xy)}.$$

For the insert and delete operations we apply a similar approach. For more details see [43].

In order to be able to use an efficient search strategy (for example: to apply pruning) the distance should be metric [30] [43] therefore we need that the distance satisfies the triangular inequality. Moreover the nature of the errors in names, implies that one edit operation cannot be substituted by two or more operations. In other terms, we need that the cost of each operation is not greater than the cost of an equivalent sequence of operations. Therefore, in order to use the discriminations as costs of the operations, we scale them in such a way that $\min(D_{op}) \geq (\max(D_{op}))/2$.

Positions

The costs we propose depend on the involved operation types and letters. To improve the efficacy we also take into account the position where the operation occur. We distinguish between the *first* position, the *last* position and the other positions, or *general* position. The probabilistic model consists of three confusion matrix (one for each position type) containing the probabilities of the $1053 = 3 \cdot (26 + ((26^2 - 26)/2))$ different operations and three vectors with the prior probabilities of the characters. To obtain the 1053 costs, we transformed the set of 1053 discriminations, in such a way to comply the triangular inequality.

Thresholds

The number of errors made (the number of edit operations) is not independent of the length of the names: two errors in a name of 15 characters are more acceptable than two errors in a name of 4 characters. Therefore, the parameter k , the distance threshold to choose the similarity degree, in the function *Similars* (x, C_l, k), should depend on the length of the name. However, the length of the two strings involved (the query string and the DB string) can be different. In some searching techniques, the value of the threshold is needed without knowing the length of the string from the DB. For example, that is the case of trie-trees. Then, we have decided to use the length of the query string. Moreover, some testing shows that the results are practically the same using the length of the DB string instead of the length of the query. And using the length of the shortest or the length of the longest, the results are a bit worse.

In order to facilitate the comparison of DEA with other functions, we have decided to define seven degrees of similarity; A,B,C,D,E,F and G. Each degree consist on a set of threshold values, one for each possible query length. We have grouped the string pairs of *TEST* by the length of the left string of the pair, and for each length the average distance is computed. The same has been done with the pairs of *CONTROL*. It seems reasonable that the threshold for each length, should be between these two average values. The concrete set of values for each one of the seven threshold degrees, have been determined after some trial-and-error.

Other Distance Functions

Through the years, a large amount of proposals have been made for the determination of the similarity of two words, based on the comparison of its characters. We comment now on three distances that we will empirically compare with the simple and DEA distances, in the next section.

Bigrams: Some very popular distances between words, are based on *n-grams*. We tested several forms of distances based on bigrams and trigrams, and the best results for names were obtained with the following distance expression:

$$\delta(x,y) = \frac{B_x + B_y - 2B_{xy}}{2B_{xy}}$$

where B_x is the number of different bigrams existing in the word x , B_y is the number of different bigrams existing in y , and B_{xy} is the number of different bigrams common to both words. When there are no common bigrams, the value of B_{xy} will be 0.5.

Jaro: To detect coincidences of people during the processes of the US Census, a distance is used based on comparisons of characters, devised specifically for surnames, that we call Jaro distance [44]. It takes into account: a) the length of the surnames, b) transpositions of characters, c) the coincidences of characters (if their positions are not separated more than the half of the length of the shorter surname), and d) the number of similar characters. Vowels and the following pairs of characters are considered similar: *BV B8 CG CK CQ EF EY Eblank GJ IJ IL IY II KX MN 0(zero) O(letter) PR QC Q0(zero) QO(letter) SX SZ S5 Sblank UV UW VW Yblank Z2*.

Editex: Zobel and Dart [13] propose a comparison function, Editex, that they tested using personal names. Editex is a variant of a weighted edit distance, where only three different costs exist: coincidence, similar and non-similar. For the insert/delete operations, the possible similarity or coincidence of the previous character is considered. The letters *H* and *W*, and the duplicates of characters, receive a special treatment. The similarity criterion is based on the phonetic groups of the PHONIX codification system [45].

3. EVALUATING AND COMPARING DISTANCE FUNCTIONS

Objective Evaluations

In this section five distances are compared: Simple distance, Bigrams, Jaro, Editex and DEA. The efficacy is related to the hits and faults in the identification of the similarity. In the area of Information Retrieval (IR), *relevance judgments* made by human judges, are used to decide if the retrieved documents are relevant or not to the query. When searching people by name, this type of evaluation procedure is not appropriate because it is too subjective and unsteady. However, it is the procedure traditionally employed in the comparison of personal-name matching methods [5], [13], [26] and [27]. For example, in [27] the judge ("professional documentalist") is asked to apply subjective criteria as: "sounds about the same", "obvious typing mistake", and "pronunciation is not affected significantly".

In order to avoid the evaluation subjectivity, we adopt an approach based on a corpus containing real errors and variants, and applying again the discrimination concept. We are interested in the empirical evaluation of how the different distance functions are able to correctly discriminate between *pairs-with-error* (a test file) and *pairs-without-error* (a control file).

We will not use the same files (*TEST* / *CONTROL*) that were used in section 2 for the determination of the DEA costs, but another pair of files *TESTR* / *CONTROLR*. The file *TESTR* is a file with 519 *pairs-with-error*, and its origin is not related with the *TEST* file. The file *CONTROLR* consists of 519 *pairs-without-error*, obtained pairing at random the left hand surnames with the right hand surnames of *TESTR*.

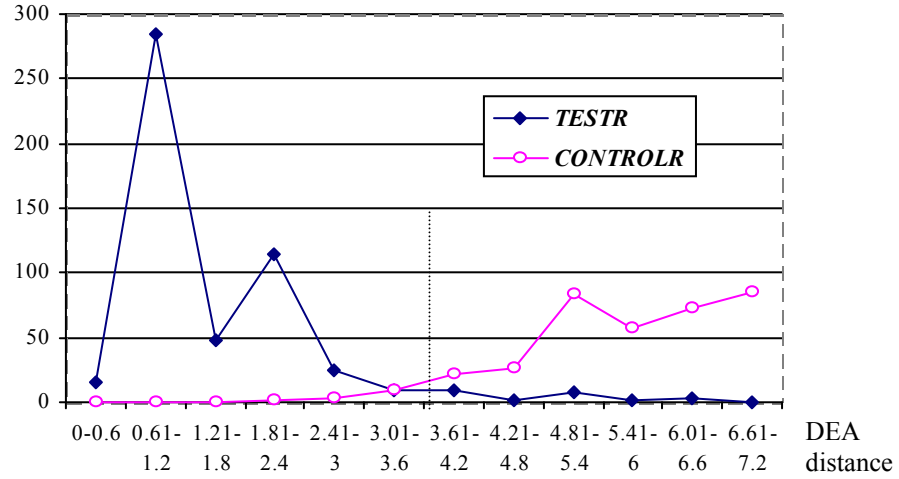


Figure 1: Distribution of the DEA distance

Figure 1 show a line for each file, displaying the frequency distribution of the DEA distance. Intuitively, the discrimination between the two files is as much better as more separated are the two lines. Now suppose that we have all the surname pairs of both files, *TESTR* and *CONTROLR*, together into a single set, and we try to identify the pairs pertaining to *TESTR*, that is, the *pairs-with-error*. If we use a distance threshold as a discrimination criterion (in the figure we suppose that the threshold is 3.61), a partition is produced in four sets as it is shown in figure 2 : a) pairs-with-error (pairs from *TESTR*) identified correctly as such, b) **false positives** pairs, that is pairs-without-error (pairs actually from *CONTROLR*) identified as pairs-with-error, c) **false negatives**, that is pairs-with-error (pairs actually form *TESTR*) identified as pairs-without-error, d) pairs-without-error (pairs from *CONTROLR*) identified correctly as such. The total number, N , of pairs of the experiment, is the number of pairs in *TESTR* plus the number of pairs in *CONTROLR*.. The table of the figure2 is called contingency table in the hypothesis tests.

	pairs-with-error (<i>TESTR</i>)	pairs-without-error (<i>CONTROLR</i>)
identified as pairs-with-error	<i>Correctly identified as pairs from TESTR</i>	<i>False positives</i>
identified as pairs-without-error	<i>False negatives</i>	<i>Correctly identified as pairs from CONTROLR</i>

Figure 2: The contingency table: partition in four sets

In the example of figure 1, the left hand area of the vertical line ($\delta = 3.61$) and below the line of *CONTROLR*, is the area of the *overidentified* pairs or *false positives*. The right hand area of the vertical line but below the *TESTR* line, corresponds to *misidentified* pairs or *false negatives*. The discrimination is as much better as fewer anomalies of both types are produced.

The metrics we will use to quantify these anomalies are very popular in the IR field: *Fallout* and *Recall*:

- The *Fallout* is the probability that a pair-without-error is a false positive. It will be noted as F . We will also use the term *overidentification* with this same meaning.
- The *Recall*, noted as R , is the probability that a pair-with-error is identified as such. Often we prefer to use its complement, named here with the term *misidentification*, defined as the probability that a pair-with-error is a false negative. So, $\text{misidentification} = 1 - R$.

In table 1, the values of F and $1 - R$ are given for the five distance functions that we are analyzing, and for several thresholds.

The *MiFa* graphic

In figure 3 we display the relationship between $1 - R$ and F for our corpus: *TESTR* and *CONTROLR*. We have called *MiFa* the graphic that relates the misidentification with the fallout or overidentification. This graphic allows choosing the more appropriate threshold for each application. In the IR field, sometimes a graphic *Recall/Fallout* is used, though the *Recall/Precision* graphic is more popular. The *MiFa* graphic is widely used (under other names) in other fields as for example in biometrics or clinical research.

Usually, in practice, misidentification values greater than 20%(approx.), and fallout values greater than 2%(approx.), are unacceptable. Therefore, in figure 3 we limit the F and $1 - R$ values to this interval. Into this interval, the simple distance function does not allow to tune the similarity criterion, the threshold, because only a single point exists, $\delta \leq 2$, since $\delta \leq 1$ and $\delta \leq 3$ are out of this interval and the simple distance is an integer. The DEA function has several points corresponding to the threshold degrees we have defined depending on the lengths (see section 2) but more points could be defined because DEA produces, within this interval, more than 100 different distances.

The methods of phonetic codification are out of this interval. Examples: The SOUNDEX [7] method has $1 - R = 46.6\%$ $F = 0.43\%$, and the NYSIIS system [8] has $1 - R = 59.8\%$ $F = 0.08\%$.

The target is to minimize both, F and $1 - R$. A look at figure 3, shows that DEA is the function that better fulfills this target. The other four functions are very similar between them. For the same level of misidentification (or recall), the DEA function gives 70% to 80% lower fallout than the other functions (within our working interval). For the same level of fallout, it gives a 40% to 55% lower misidentification.

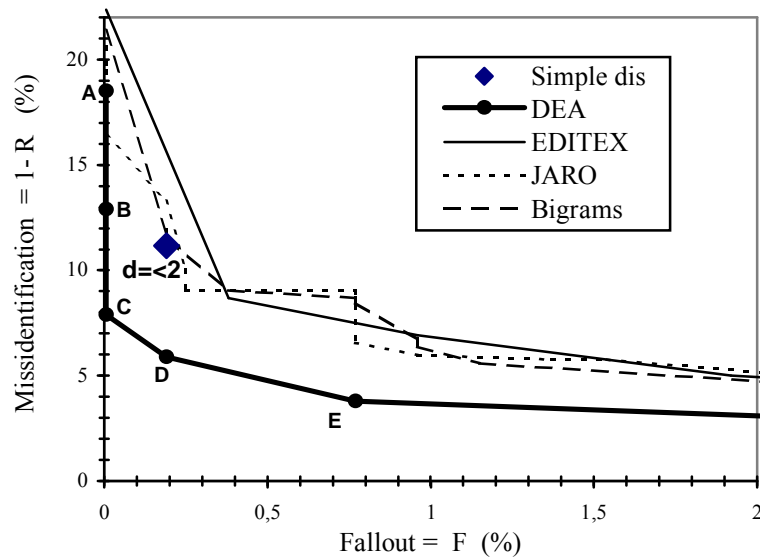


Figure 3. Graphic *MiFa* for *TESTR/CONTROLR*

Table 1: Evaluation of the distance functions (values in %)

Function	DEA			JARO		BIGRAM	EDITEX	SimpleDis
Threshold	C	D	E	0.14	0.18	1.1	6	2
Misid. $1-R$	7.9	5.9	3.8	13.29	6.55	11.94	8.67	11.17
Fallout F	0.005	0.19	0.77	0.19	0.77	0.19	0.38	0.19
Prec. $P \quad \beta=1$	99.99	99.79	99.20	99.78	99.18	99.78	99.58	99.78
Efficacy $E \quad "$	96.05	96.96	97.72	93.26	96.34	93.94	95.48	94.32
" $J \quad "$	95.89	96.87	97.68	92.79	96.23	93.56	95.28	93.99
Prec $P \quad \beta=10^{-3}$	94.85	33.12	11.11	31.33	10.82	31.67	19.38	31.85
Efficacy $E \quad "$	99.98	99.80	99.22	99.79	99.22	99.79	99.61	99.79
" $J \quad "$	93.46	48.99	19.91	46.03	19.4	46.59	31.97	46.89

Data Volumes, β -Factor and Precision

With the values of R and F alone, it is not possible to compute the volumes of the four sets of the partition produced by a threshold. Therefore in order to predict these volumes, for example the number of false positives, we will also use N , the total number of pairs, and a factor β , expressing the ratio between the number of pairs-with-error and pairs-without-error. For most of applications involving personal-names, this β factor will have values lower than 0.01 and very often lower than 0.001. Now we can express the number of non-desired answers (the false positives) as a function of N , F and β , by :

$$N \frac{F}{\beta + 1}$$

See that for high N values and low β (these are the usual conditions) the number of not-desired answers is very high, although the F value can look very low.

It can happen that the misidentification and the fallout are both rather low, but the ratio of false positives is very high. This can be unacceptable for many applications, because of psychological reasons or the difficulty of handling the answer. Therefore, it may be useful to use the *Precision* metric, P , very common in IR. We can define the precision as the probability that a pair identified as a pair-with-error, is really a pair-with-error. So:

$$P = \frac{\beta R}{\beta R + F}$$

Note that if the proportion β of names in the DB similar to the searched one descends, then the precision will decrease.

Example

We will analyze here the behavior of the distance functions for a hypothetical DB containing records for 4 million people. To access the personal records using the first surname, we build a directory with all the approximately 100,000 different first surnames existing in the DB for these people. Suppose that the correct answer to our query in the directory should have 100 surnames. Then the β factor is $\beta = 100/(100,000-100) \approx 0.001$.

Now suppose that we do not accept misidentifying more than 5 surnames, that is, we need $R > 95\%$. And we do not accept in the answer more than 1500 false positives, that is, $F < 1.5\%$. From figure 3 we see that there is no other solution than the DEA function with the threshold degree E. This point has a recall $R = 1 - 0.038 = 96.8\%$ and a fallout $F = 0.77\%$ (table1).

With the DEA function and the threshold degree E (DEA-E), we can expect a total answer of 865 ($=N(\beta R + F)/(\beta + 1)$) candidate surnames, from which only 96 can be anomalies of the searched surname. The other 769 surnames of the answer, are false positives. Therefore, the precision P is very low, $P = 11.1\%$, and in the answer 4 surnames are missed (false negatives). All these resulting values fall within the limits we have imposed to R and F .

If we use the simple distance with the threshold $\delta \leq 2$, we obtain a misidentification of 11.17% (11 false negatives) that is more than the double of the imposed limit. But the fallout value is as low as 0.19%. And using the Bigrams distance with a threshold ≤ 1.1 we obtain similar results. Using the Jaro distance with the threshold 1.8 we obtain the same fallout, $F = 0.77\%$, than with DEA-E, but the misidentification is 6.55%, almost the double of DEA-E, and off the imposed limit.

If the fallout $F = 0.77\%$ (769 false positives) is too high for our needs, and we want to reduce it to $F = 0.19\%$ (the same value than it has in Bigrams-1.1, Jaro-1.4 or simple distance $\delta \leq 2$) we can use DEA but with the D threshold degree. Now we obtain only 190 not-desired surnames, but the misidentification has moved from 3.8% to 5.9%. That is a bit over the imposed limit, but it still is much below the value obtained with the simple distance, Bigrams-1.1, or Jaro-0.14. For DEA-D and $\beta = 0.001$, the precision is $P = 33.12\%$.

For some applications, the fallout of DEA-D could still be too negative from a psychological point of view. Probably the user will not accept in the answer, surnames that he/she does not consider similar to the query. But the misidentification can go unnoticed because the user is not knowledgeable of which valid surnames have not been given in the answer. If the fallout of DEA-D is still considered excessively high, there still are more strict threshold degrees; A, B and C. With these threshold degrees the precision is greater than 90%, but the misidentification is then greater than 7%.

Two surnames

Until now we considered the use of only one surname, but in Spain the norm is to have (and use) two surnames. We want to find all the people in the DB, having their two surnames similar to (or able to be confused with) the two surnames we are searching.

The ratio between the amount of people we are interested in, and the amount of people we are not, the new beta factor, will be labeled as β_2 , and is:

$$\beta_2 = \frac{\beta^2}{2\beta + 1}$$

For $\beta < 0.1$ (it's almost always the case) the new factor value is $\beta_2 \approx \beta^2$

The recall for the two surnames case is $R_2 = R^2$. Therefore, the recall will descend. The new fallout is:

$$F_2 = \frac{2\beta FR + F^2}{2\beta + 1}$$

Note that F_2 is not only depending on the threshold (this is, on F and R) but also on β , because some of the overidentified people have one of the two surnames correctly identified as similar. The fallout also descends. For the usual values of β , the new fallout is $F_2 \approx F^2$.

Table 2 shows the main results for the two surnames case.

The misidentification ($1-R$) for two surnames is twice (approximately) the value for one surname. DEA-E has a recall, for two surnames, of 92.54% (96.2% for one surname) that is, a misidentification of 7.46% (3.8% for one surname). If this recall value is not good enough for our application, we can raise it by using a more tolerant threshold degree; using DEA-G the recall is 96.57%.

Table 2: Evaluation of the distance functions, for two surnames (values in %)

Function		DEA			JARO	BIGRAM	EDITEX	Simple dist
Threshold		E	F	G	0.32	3	1.2	4
(one surname) $1-R$		3.8	2.7	1.73	1.9	1.56	2.3	2.5
" " F		0.77	2.7	3.86	6.55	5.39	14.45	10
$1-R_2$		7.46	5.327	3.43	3.76	3.056	4.54	4.94
$\beta=10^{-3}$	F_2	0.007	0.078	0.156	0.441	0.3	2.11	1.017
	P_2	1.23	0.121	0.061	0.021	0.032	0.004	0.009
	E_2	99.97	99.87	99.77	99.45	99.61	97.67	98.82
	J_2	2.439	0.242	0.123	0.043	0.064	0.009	0.018

The fallout is lower than the case of only one surname, but now it depends on the value of β . If the value of β for only one surname is 0.001, then $\beta_2 \approx 0.000001$ and the new fallout for DEA-G is $F_2 = 0.156\%$. With such a small value of β_2 , the precision falls down terribly: P_2 is now under 1%. In order to obtain a precision $P_2 \geq 50\%$ we need to use the A, B or C threshold degrees, but then the recall is under 90%.

In table 2 we see that for two surnames, the DEA function gives better results than all other functions: for similar values of recall, a much lower fallout is produced.

The Recall/Precision graphic

In the IR field, in order to choose a criterion of document selection, a *Recall/Precision* graphic is usually used instead of the *MiFa* graphic. For a given recall level, if β decreases, the precision decreases very quickly. In figure 4 we display for $\beta = 0.001$ the *Recall/Precision* graphic for the files *TESTR/CONTROLR*. The lines for the Bigrams and Editex functions are not displayed because their behavior is nearly the same than the Jaro function (see table 1).

To obtain, with $\beta = 0.001$, a precision greater than 50%, we need to accept a recall R lower than 95%. To obtain precision values greater than 50% using the simple distance, we need a threshold $\delta=1$, but that produces a terribly low recall (58.56%).

The tension between P and R , grows when the two surnames are used. To obtain P_2 values about 50%, we need to accept a recall lower than 90%. Obviously, with greater β values the precision is also greater. For example; using DEA-E and having $\beta = 0.01$ we obtain a $P_2 = 55\%$ and a recall of 96.2%, but these β values are not very usual in practice.

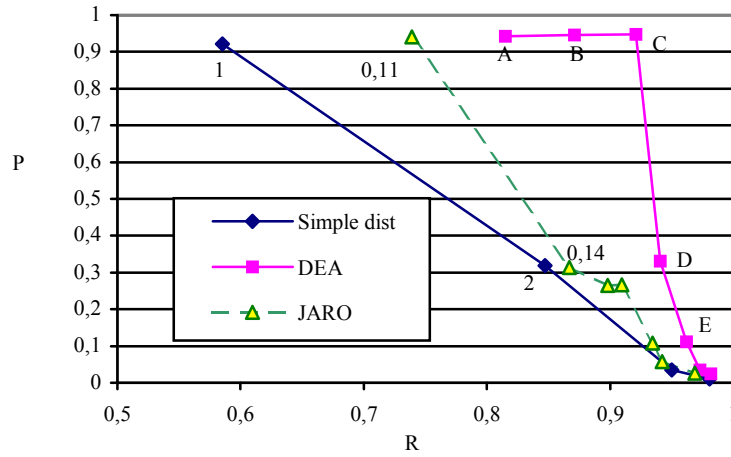


Figure 4: Recall/Precision graphic for $\beta=0.001$

Efficacy: E and J

We have seen two basic ways to evaluate the goodness or efficacy of a function: One is based on the pair R/F (the *MiFa* graphic) the other is based on the pair R/P (the *Recall/Precision* graphic). With any of these two pairs we can select the best threshold for our needs. Overlapping several graphics for different distance functions, we can choose the most appropriate distance for our problem.

It looks as a natural desire to be able to evaluate the efficacy handling a single metric instead of a pair. In several fields where contingency tables are used, single metrics are defined with the name of *Efficacy* or *Effectivity*. These metrics are not easy to use and interpret [46] [47]. The efficacy needed for an application is more naturally expressed by means of *Recall*, *Fallout* or *Precision*. In the IR field, when different methods are to be compared, the average of the precision P for several R values is often used. This requires producing the same R values for all the methods. But this is usually impossible in the name-matching problem.

We present in the following paragraphs the two more common metrics of efficacy used in the IR world: A combination of the pair R/F , and a combination of the pair R/P .

We name as *Efficacy* E , the probability to be right in the identification of the type of a pair of names, including the two types of hits; *pair-with-error* and *pair-without-error*. So:

$$E = \frac{\beta R + (1 - F)}{\beta + 1}$$

As can be seen, E is the average between R and $1-F$, weighted with β . Then, $E_{\beta=0} = 1 - F$ and $E_{\beta \rightarrow \infty} = R$. In other words, $1-F$ and R are the limits of E when β is varying. When $\beta < 0.01$ the values of E are almost equal to $1-F$.

We can have too many false-positives, a low precision, in spite of having a very good value of E . Because of this, another type of efficacy metric can be convenient related with P . In the IR field the metric defined by Jardine and Rijsbergen [48] [49] under the name of *Effectivity*, is very usual. More recently it is being used also in computational linguistics [29]. We use here the letter J to refer to this metric and will be expressed as:

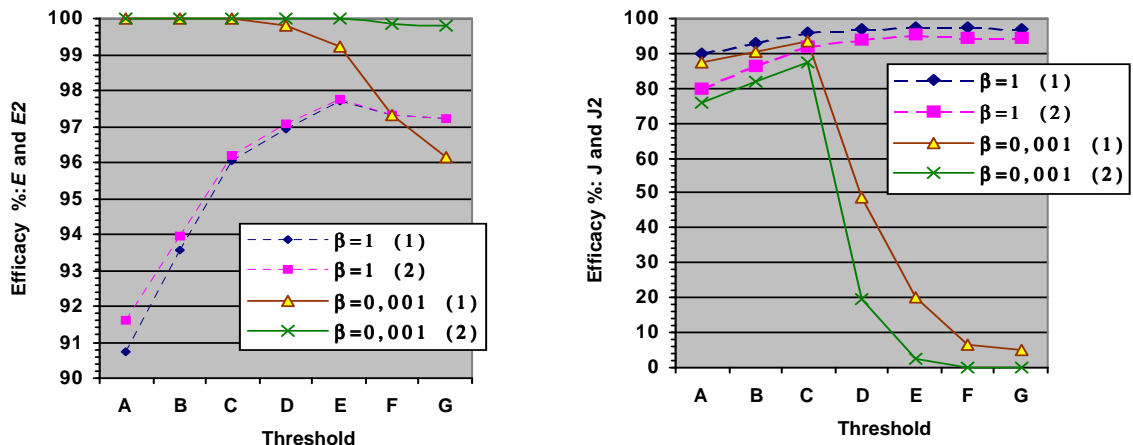
$$J = \frac{2PR}{R+P}$$

that is, the harmonic average between the precision P and the recall R , two values that we need to be as high as possible.

We limit ourselves to comment here, with the help of graphics, some of the values of E and J obtained with the files *TESTR* / *CONTROLR* (see tables 1 and 2). In the figure 5a we represent the E of the DEA distance, as a function of the threshold for one and two surnames, and for $\beta = 1$ and $\beta = 0.001$. As E is the average between R and $1-F$, weighted by β , for small values of β we obtain $E \approx 1-F$. So, if we use a very small F , a large E is obtained. As we work usually with $\beta \leq 0.001$ and with a small F , the efficacy E is over 95%. And if two surnames are used the efficacy is over 99%.

For the case of only one surname, the DEA-F threshold degree always produces the same efficacy E , no matter which is the value of β . A high efficacy E does not prevent a very low precision. For example; for DEA-E with $\beta = 0.001$, we obtain $E = 99.80\%$ but the precision is only $P = 33.12\%$. If two surnames are used, the efficacy is even higher, $E_2 = 99.97\%$ but the precision falls down to $P_2 = 1.23\%$. With any value of β , it is $E_2 > E$ and $J > J_2$. So, using the two surnames, the efficacy E (based on F/R) increases, while the J (based on P/R) decreases.

In the figure 5b we represent the J of the DEA distance, as a function of the threshold degree.



(1) = One surname (2) = Two surnames

Figures 5a and 5b : The efficacy E and the effectivity J , for the DEA function

The single metrics of efficacy or effectivity, E or J , are not useful to compare different distances having different thresholds. We need to substitute the thresholds by a metric as R or F . As can be seen in tables 1 and 2, if we compare the E or J values for similar values of R or F , the DEA method always has higher values than the other functions analyzed here.

Summary of the comparison between distance functions

Using a non-subjective evaluation method (unlike what is usually done) we have compared five approximate matching functions being used for searching personal-names. The simple edit distance is not an appropriate function. With the threshold $\delta \leq 3$ too much fallout is produced. With the threshold $\delta \leq 1$ too many false negatives (too low recall) are obtained. The threshold $\delta \leq 2$ still produces more false negatives (recall lower than 89%) but in some circumstances it can be accepted. No intermediate thresholds are possible.

The other functions compared in this work, except DEA, are not significantly better than the simple distance. But the DEA function gives us important improvements, because the costs of the edit operations depend on the involved letters and their position, and the distance threshold is variable. For the same level of recall, the DEA function gives a fallout from 70% to 80% lower, and for the same level of fallout, it gives a misidentification from 40% to 55% lower.

When the two surnames are used, similar results are obtained, DEA is again producing higher values of E , J , E_2 and J_2 , for similar values of F or R .

With the usual values of β , DEA-C is the threshold degree of choice for most circumstances.

The efficacy of known phonetic codification methods, is significantly lower than the efficacy of distance functions [43].

4. SEARCHING TECHNIQUE

Object

In the previous section we have presented the DEA function, and showed that it has higher efficacy than the other functions we compared with. This answers only one of the two APNM questions presented in section 1: *which similarity criterion is to be adopted?* But now we need an answer to the second question: *how to search, using such similarity criterion?*

We need an efficient search technique, that allow us to find in a DB all the personal names that are similar to the query. Efficient, means fast enough; perhaps we need a response time below few seconds in spite of having a high volume DB and a big amount of simultaneous users. We need a search strategy in a data structure, and an algorithm to assess the similarity.

If the IS has a high volume DB with many users, the sequential search, looking one by one all the personal names, is too much inefficient, so we need a more sophisticated technique having better performance. But distance functions, are still resisting really efficient filters or index techniques [50]. In this section we propose a very efficient approximate search technique for the DEA function, based on a trie-tree.

Basic algorithm for edit distances with costs

Lots of papers have been published proposing efficient algorithms to calculate the simple distance. In the Navarro survey about ASM [22] only algorithms for the simple distance are included. Remember that simple distance means that every operation has a cost =1. The research on efficient techniques for more general costs is still in its first steps.

As we know, the edit distance between two strings can be defined as the minimum cost of all the possible sequences of edit operations that transforms one string into the other. The cost of a sequence is the sum of the costs of their operations. The costs of the elementary operations, are non-negative real numbers that we noted in section 2 as:

$$\begin{array}{ll} \delta_c(\epsilon, x) & \text{cost to insert the character } x \\ \delta_c(x, \epsilon) & \text{cost to delete the character } x \\ \delta_c(x, y) & \text{cost to substitute } x \text{ by } y \text{ (for } x \neq y) \end{array}$$

Remember that $\delta_c(x, \epsilon) = \delta_c(\epsilon, x)$ and $\delta_c(x, y) = \delta_c(y, x)$.

Almost all the algorithms proposed for the calculation of distance functions, are derived from the simple distance algorithm from Wagner and Lowrance [36]. This very popular algorithm is based on dynamic programming, so we will name it as DP algorithm. It consists of a recursion on a

matrix; the DP matrix. If x and y are two strings, and m and n are their respective lengths, then the DP matrix has $m+1$ rows and $n+1$ columns.

See the example of figure 6, with the DP matrix for the calculation of the distance between $x=AVERY$ and $y=GARVEY$. Suppose that all the elementary costs are equal to 1, except for the insert/delete of A , G and R , whose costs are:

$$\begin{aligned}\delta_c(A, \epsilon) &= \delta_c(\epsilon, A) = 1.1 \\ \delta_c(G, \epsilon) &= \delta_c(\epsilon, G) = 1.3 \\ \delta_c(R, \epsilon) &= \delta_c(\epsilon, R) = 1.08\end{aligned}$$

		$y_j \rightarrow$	G	A	R	V	E	Y
$i=0$	$\downarrow x_i$	0	1,3	2,4	3,48	4,48	5,48	6,48
$i=1$	A	1,1	1	1,3	2,38	3,38	4,38	5,38
$i=2$	V	2,1	2	2	2,3	2,38	3,38	4,38
$i=3$	E	3,1	3	3	3	3,3	2,38	3,38
$i=4$	R	4,18	4,08	4	3	4	3,46	3,38
$i=5$	Y	5,18	5,08	5	4	4	4,46	3,46
		$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
			min=1	min=1,3	min=2,3	min=2,38	min=2,38	min=3,38

Figure 6: DP matrix for a distance with costs

We suppose that the transformation is from x to y , so we calculate the minimum cost to transform $AVERY$ into $GARVEY$. Moving from one cell to the cell below it (incrementing i) represents a delete operation. Moving a position from a cell to the right, represents an insertion (incrementing j) and the diagonal move from the cell $(i-1, j-1)$ to the cell (i, j) represents a substitution if $x_i \neq y_j$. The distance is the cost of the less expensive path from cell $(0,0)$ to cell (m,n) . In figure 6 the set of gray cells is the less expensive path. In each cell, the minimum cost from $(0,0)$ to it is displayed. In the example, the distance between the two names is 3.46 because this is the value of the final cell $(5,6)$.

The calculation starts with a zero cost from the cell $(0,0)$ and is advancing column by column, and into each column row by row, assigning to each cell the minimum cost to reach it. This recursive algorithm to fill the DP matrix can be expressed as follows:

$$\begin{aligned}\delta(0,0) &:= 0 \\ \forall i \in \{1..m\} \quad \forall j \in \{1..n\} \quad \left\{ \begin{array}{l} \delta(i,j) := \min \left(\delta(i-1, j-1) + \delta_c(x_i, y_j) , \right. \\ \qquad \qquad \qquad \delta(i, j-1) + \delta_c(\epsilon, y_j) , \\ \qquad \qquad \qquad \delta(i-1, j) + \delta_c(x_i, \epsilon) \end{array} \right\}\end{aligned}$$

For example, let's see the calculation of the value for the cell $(2,3)$:

$$\begin{aligned}\text{Move from } (1,3) \text{ to } (2,3) &= \text{Value of cell } (1,3) + \delta_c(V, \epsilon) \rightarrow 2.38+1 = 3.38 \\ \text{Move from } (2,2) \text{ to } (2,3) &= \text{Value of cell } (2,2) + \delta_c(\epsilon, R) \rightarrow 2+1.08 = 3.08 \\ \text{Move from } (1,2) \text{ to } (2,3) &= \text{Value of cell } (1,2) + \delta_c(V, R) \rightarrow 1.3+1 = 2.3\end{aligned}$$

The value of the cell $(2,3)$ is then 2.3 because it is the lower value of the three calculations. Observe that this value is the lower of its column ($j=3$) but this cell is not in the minimum cost path. (For clarity of the figure, the minimum value of each column is repeated at bottom). Note

that several paths with the minimum cost can exist. But here we are interested in the distance (the cost), not in the details of the sequence of edit operations (the path). The amount of cells to compute, the number of iterations of the recursion, is $n*m$. In other words, the temporal complexity of the algorithm is $O(n*m)$.

Let's imagine a directory or vocabulary with the N different names existing in the DB. As can be seen in [43] when a single surname is considered, the number N of different spanish surnames as a function of the number of people, P , follows a Zipf-Mandelbrot law [51]:

$$N = 41 P^{0.501}$$

Because N is much smaller than P , we will do the approximate search in the vocabulary. We can do a sequential search, computing for each surname y its distance to x . They are similar if $\delta(x,y) \leq \text{threshold}$. With the above basic DP algorithm, the total number of columns to compute, C , is:

$$C = n*N \quad \text{where } n \text{ is the average length of the surnames of the vocabulary}$$

Since $n = 7$ approximately, the number of columns to be computed is $C = 287 P^{0.501}$

The DEA function is an edit distance function, $\delta(x,y)$ with costs. Therefore we can use the basic algorithm above. But the costs depend on the symbols and the position (first, general and last). Let's see how to compute the DEA distance between the string *DEC* and the string *BCTR*. Suppose that the costs of the operations are (these are not the actual DEA costs):

- Delete a *D* in 1st position: 0.5
- Substitute an *E* by a *B* (or a *B* by a *E*) in a position other than 1st or last: 0.6
- Insert a *T* in a position other than 1st or last: 0.65
- Insert an *R* in last position: 0.55
- All other operations: 1

In figure 7 we show the DP matrix for our example. We see that $\delta(DEC, BCTR) = 2.3$. This cost is produced by a delete of a *D* in 1st position, plus a substitution of *E* by *B* in 2nd position, plus an insert of *T* in 3rd position, plus an insert of *R* in last position. We must clarify what we mean by "position" of an operation. For an insert, the position is the position of j . For example, in figure 7 the insert of the *R* occurs in the last position, and the insert of *B* occurs in 1st position. In a delete, the position is the position of i . In a substitution, the position is the 1st if $i=j=1$ and is the last if $i=m$ and $j=n$. For example; the cost of the substitution of *R* by *E*, is not the cost of the last position but the cost of the general position.

		$y_j \rightarrow$	B	C	T	R
$i=0$	$\downarrow x_i$	0	1	2	2,65	3,2
$i=1$	D	0,5	1	2	3	3,55
$i=2$	E	1,5	1,1	2,1	3	3,55
$i=3$	C	2,5	2,1	1,1	1,75	2,3
		$j=0$	$j=1$	$j=2$	$j=3$	$j=4$
			min=1	min=1,1	min=1,75	min=2,3

Figure 7: DP matrix DP for the DEA distance

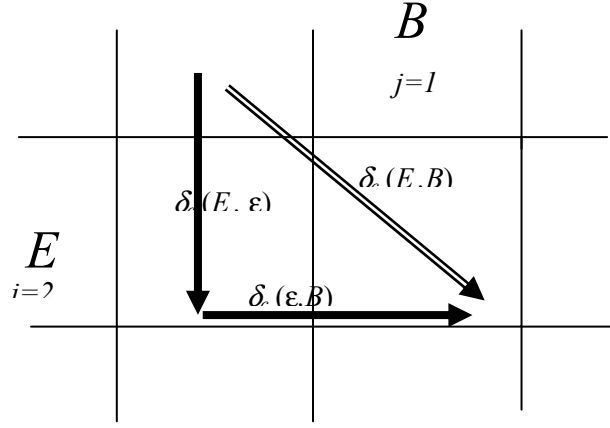


Figure 8: Triangularity

As we said in section 2, in order to be able to compute the edit distance, a sequence of edit operations cannot operate twice on the same character. If the triangularity is not observed, the DP algorithm can accept an insert followed by a delete of the character inserted (or insert a character in the position where a delete is just done). For example, in figure 8 if

$$\delta_c(E, \epsilon) + \delta_c(\epsilon, B) < \delta_c(E, B)$$

then the DP algorithm will adopt the cost of a delete plus the cost of an insertion.

Improving the basic algorithm: cut-off column

To reduce the number of cells of DP matrix to be calculated, some techniques in the context of ASM [52] [22] [50] have been proposed, but they are usually based on properties that stand when the elementary costs are all equal, as in the simple distance, but this is not the case of our DEA costs. For example, going through the diagonals of a DP matrix, when the cost is unique, you find non-decreasing values. But we see that this is not the case in figures 6 or 7.

A feature of the DP matrix is that the minimum value of the cells of every column is never smaller than the minimum of the previous column [53]. We can use this feature to reduce the number of cells to be calculated. In our case we are not interested in knowing the distances between strings, but in knowing if two strings are similar or not, given a threshold. Therefore, if while we are calculating the DP matrix cells, column by column, we find a column with a minimum value greater than the value of the threshold, we can already affirm that the two strings are not similar, so it's not necessary to continue calculating the matrix for that pair of strings. This cut-off technique produces a big saving of columns to be calculated as we will see later.

If the vocabulary is structured as an ordered list of names, we can reuse the common prefix with the previous string to save column calculations. If we are searching the pattern *BACI* in the following list $\{ABCD, ABCE, ABCEF\}$, then when we will calculate the matrix for *ABCE* we can reuse the calculation of the first three columns of the previous matrix. However, to calculate the *ABCEF* matrix we cannot reuse the *E* from *ABCE* as a possible substitution of the *I* from *BACI* because it will have been calculated with costs of the last position and it has to be recalculated with the costs of the general position.

Let's suppose that while comparing *BACI* with *ABCD*, when the second column is calculated we detect that the two strings are not similar, so the second column is the cut-off column. Then, when the matrix for the string *ABCE* is calculated, only two columns can be reused, the third one not being calculated yet because of the cut-off.

Improving the search: trie-tree

Till here we have assumed that the approximate search is done by a sequential access to the names in the vocabulary, structured as an ordered list, and comparing each name with the pattern string.

The research about ASM algorithms has been strong from the 80s, but until recently, efficient preselection/filtering/indexing techniques have not been incorporated. For example, the first global vision of indexation techniques applied to ASM, has been published in 2001 [50], and it is almost exclusively oriented to simple distance.

We need to have a structure for the vocabulary that allows an efficient approximate searching applying the DP algorithm improved with the cut-off column. We selected the trie-tree as an appropriate structure [7] [54] [55] [56] [50].

The trie-tree is a structure where the common prefixes are not repeated: there is a node for each common prefix. When an exact search is done in a trie-tree, the amount of nodes to read is always equal to the length of the pattern string, and therefore it is independent of the number of strings. It is the most used structure for the exact search in dictionaries or vocabularies of text strings.

In our trie-tree, each leaf corresponds to a name of the vocabulary. The column-by-column advancement in the DP matrix, is now an advancement from the root to the leaf corresponding to the name being compared with the pattern. The advancement from one name to the next, is done by *backtracking*, this is, in preorder. The trie structure allows two important efficiency improvements: a) avoid the calculation of the columns corresponding to common prefixes, in addition to save their space, and b) avoid the calculation of the columns of the subtrees when the cut-off column is reached. Our approach is very similar to the method presented on a recent paper about Matchsimile [28].

Let us see an example. We have a vocabulary with the 11 following names (containing 47 letters):

{ *ANA, CAMPO, CAMPON, CAMPONA, CAMPS, CAMS, CEL, CELIA, CELO, DO, DON* }

In figure 9 we represent it as a trie. The last letter of each name is signaled with a [] symbol. There are:

- 20 characters (instead of 47)
- 20 edges (or pointers)
- 11 *leaves* (the last letters of the names)
 - 7 being *terminal leaves* (as the final *A* from *ANA* or *CAMPONA*)
 - 4 being *intermediate leaves* (as the *O* from *CAMPO* or the *L* from *CEL*)

When searching in the *trie*, after the determination of the similarity between a name and the search pattern, we will continue to determine the similarity of the next name. As this name can have some of its first characters equal to the previous (the common path from the root) we can avoid calculating the columns of the DP matrix for these characters.

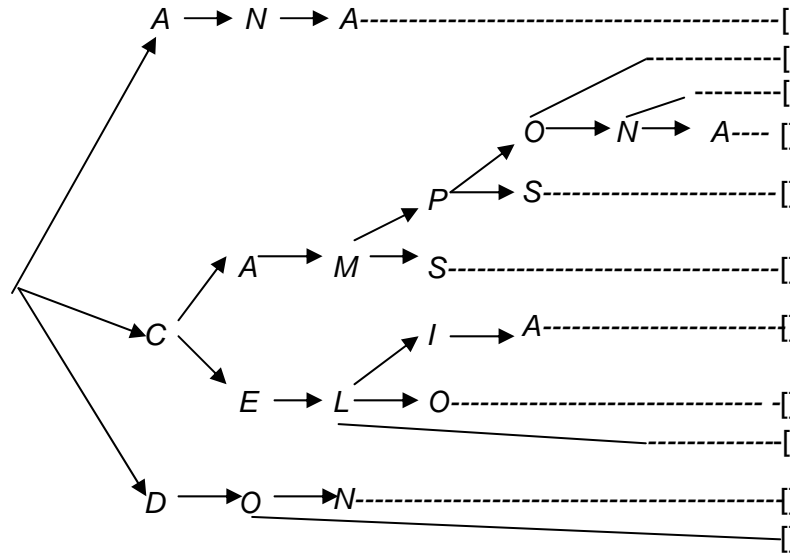


Figure 9: Trie- tree

Let us see in the example, how a trie-tree can help us reduce the number of columns to be calculated. We calculate columns of the DP matrix, advancing through the tree from the root to the leaves. Let us suppose that we have already finished the calculation of the DP matrix for the pair *pattern/ANA* and we go to the next name, *CAMPO*, so we keep going through the path *root*→*C*→*A*→*M*→*P* at the same time that we calculate the corresponding columns. When studying the letter *P* we see that it is the cut-off column (if its minimum value exceeds the threshold) so we deduce that *CAMPO* is "not similar" to the pattern. Now we can skip the analysis of *CAMPON*, *CAMPONA* and *CAMPS* since all these names will be "not similar" because they have the common prefix *CAMP*. Therefore, we can skip the subtree after the *P* and reuse the calculations we have already done for *CAM*. Now we calculate the *S* column from *CAMS*. When finishing with the *S* from *CAMS*, either the result is "are similar" or is not, we will go on to analyze the following name, *CEL*, going directly to the *E* because the *C* is reused. But after analyzing the *L* from *CEL* we cannot reuse that column for *CELIA* because the *L* from *CEL* is in final position (*L* is leaf) and the costs in DEA function for last position are different to the other positions. We must recalculate the last column. After the analysis of the *A* from *CELIA* we will be able to reuse *CEL* to analyze the *O* from *CELO*. And after *CELO* we cannot reuse anything and we will go on to the *D* from *DON*. Etc.

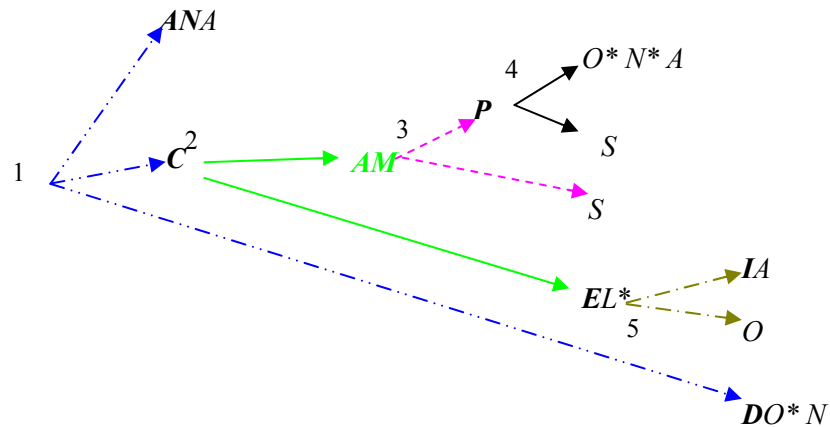


Figure 10: Compacted *trie*-tree

In order to have an efficient search we have to try keeping the vocabulary resident in internal memory. For this to be possible we will use a more compact trie-tree structure. There are several techniques for compacting trie-trees, some are very sophisticated [57] but the most of them

affect considerably the performance of the search algorithm. In the implementation that we have chosen here, we do a very simple compaction: every character with only one edge is grouped with the next one. In our example, the trie-tree will be reduced as seen in figure 10. Now it will only have 11 edges, instead of 20.

We adopt a physical representation in memory, figure 11, where there are groups of pointers and groups of characters. Every group of pointers is a node. Nodes and characters follow one another in the same order that we will use them during the search.

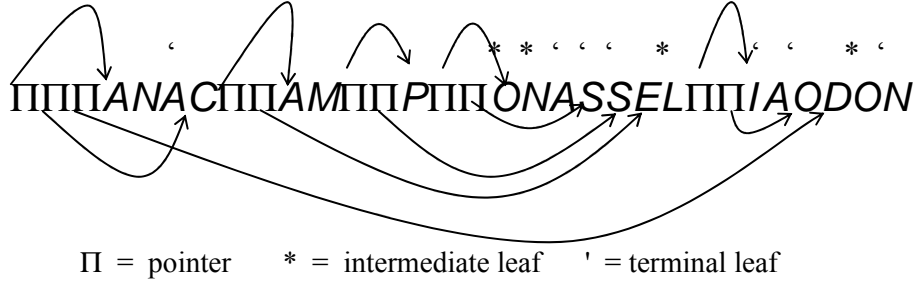


Figure 11: Structure of the *trie* tree in memory

In our implementation, the pointers are of variable length: from 1 to 4 bytes. The two first bits of every pointer show its length. Therefore, pointers of 1 byte allow 64 values, and the pointers of 4 bytes allow $2^{30} = 1,073,741,824$ values. But in all the *tries* built in the tests explained a little bit afterwards, whatever their volume is, more than 97% of pointers have a size of only one byte. The pointers of more of 3 bytes are really exceptional. The average length of pointers is 1.02 bytes. The signals that indicate if a character is leaf and if it is intermediate or terminal, are in the two first bits of the byte of the character. With the 6 bits left we can represent an alphabet of 64 symbols.

In the small example in figure 11, the vocabulary that initially had 47 characters, now occupies 31 bytes: 20 letters and 11 pointers (obviously all them of only one byte).

We have built *tries* for some files (vocabularies) of surnames:

- Five files of different sizes taken randomly from a file, *APELLIDOS*, that contains 74112 different surnames (correspond to 1.6 millions of Spaniard). Every file has in its name, the amount of surnames that it contains: A-1853, A-3705, A-18523, A-37045, A-73724
- Two files taken randomly from the file *INTERNA*, that has almost 300.000 different surnames from the entire world: I-74391, I-148781, I-297398
- A file of North American surnames, *USAlast*: U-31918

The results obtained can be seen in table 3 and in figure 12. In the figure we show the volume of the *trie* depending on the number of surnames N . Note that the graphic is doubly logarithmic.

With the most usual values of N , the amount of bytes occupied by a *trie* is between 35% and 50% lower than the list of surnames. A vocabulary of 74000 Spanish different surnames occupies, with the structure *trie* adopted here, only 292Kbytes.

Table 3: Some figures about the trie-trees

File	cList	cTrie/cList	bTrie/cList	BPun/bTrie
A- 73724	549244	37.05 %	52.94 %	30.00 %
A- 37045	276396	44.22 %	61.41 %	27.98 %
A- 18523	138450	50.22 %	67.95 %	26.07 %
A-3705	27881	62.12 %	80.01 %	22.35 %
A-1853	13751	66.14 %	84.46 %	21.67 %
I- 297398	2195273	35.43 %	51.11 %	30.68 %
I-148781	1099038	42.28 %	59.12 %	28.45 %
I-74391	2195273	47.82 %	65.21 %	26.66 %
U- 31918	220614	44.73 %	62.49%	28.44 %

cList = number of characters of the vocabulary if it was a list

cTrie = " " " in the *trie*

bPun = " " bytes occupied by pointers

bTrie = " " " of the *trie*

For a database with 20 million Spaniards, all of them with two surnames, the size of the vocabulary *trie* is around 900 Kbytes.

In figure 11 we saw that the trie-tree contains groups of pointers and groups of characters, alternately. Every group of pointers corresponds to a node. Every group of pointers is followed by the characters to go through, till the next group of pointers. The groups of pointers and the characters are sequenced in the same order used during the search.

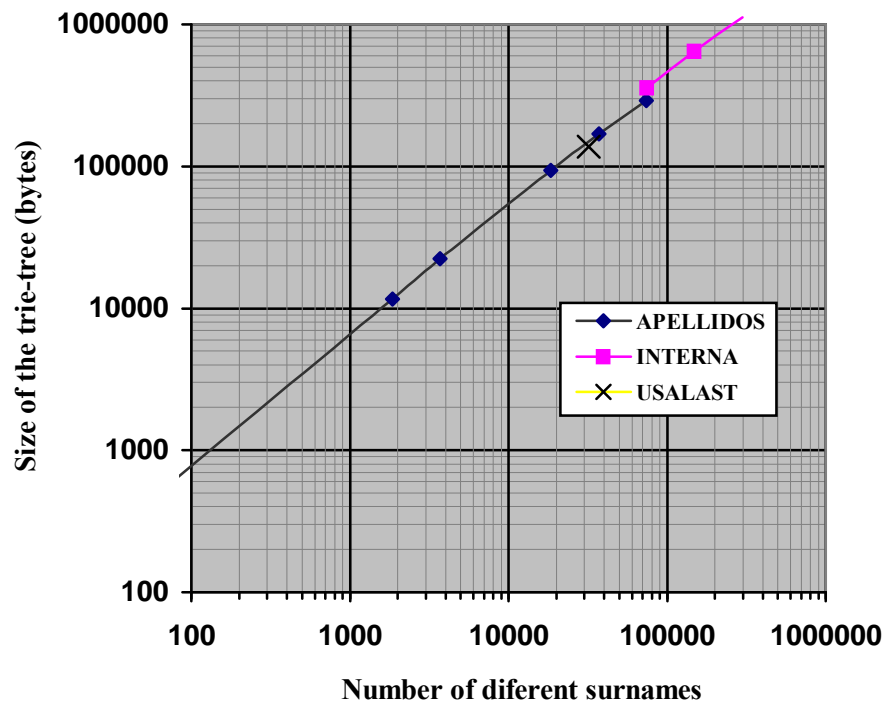


Figure 12: Size of the trie-tree

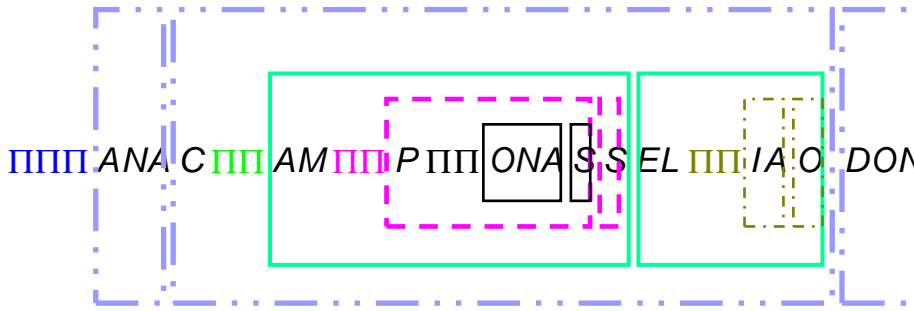


Figure 13: Nesting of subtrees in the trie-tree of figure 6

In figure 13 we have drawn as rectangles the subtrees pointed out by each pointer, Π (see also figure 11). After the first node, there are the three rectangles pointed out by the three pointers of this node. Inside the second rectangle, after the letter C (the character of the node and root of this subtree) appears the second group of pointers. And after it, two rectangles that correspond to the two pointers of this node. Etc. Our algorithm goes through this nesting in a recursive way. A detailed description of the algorithm is given in [43]

Evaluation of the efficiency

In order to experimentally analyze the efficiency of the DEA search technique, we use the files obtained from *APELLIDOS*, *INTERNA* and *USAlast* presented above (see table 3). As search pattern we use ten Spanish surnames randomly chosen.

The platform used for the test is a *Sun Sparc Ultra60* of 360MHz, 128Mb, a *SCSI* disc of 7200 rpm and the *Solaris 2.6* operating system.

We have done 700 searching tests corresponding to the ten chosen surnames for each one of the seven degrees of threshold (A to G) for each one of the ten files. We give here some of the times obtained. The time is the *user-time* given by *Solaris* and we express it in seconds (the system precision is limited to 10^{-2} seconds).

	<u>min</u>	<u>average</u>	<u>max</u>
<i>APELLIDOS</i> , A-73724 :			
Threshold degree A	0	0.021	0.06
" D	0.02	0.061	0.13
" G	0.08	0.184	0.29
<i>INTERNA</i> , I-297389 :			
Threshold degree A	0.01	0.055	0.13
" D	0.07	0.183	0.40
" G	0.27	0.646	0.97

The most expensive case, the search in I-297389 using the threshold degree G, is consuming less than 1 second. Using the D threshold degree and the A-73724 file (the first surnames of 1.6 millions of Spaniards) a search time of 0.061 seconds is obtained. These times are good enough to allow the efficient use of our algorithm in IS that demand high performance.

Now we will analyze the number of operations done on the *trie* by our algorithm, depending on the vocabulary size. The number of operations is linearly dependent on the number C of characters analyzed. And C is equal to the number of DP columns calculated. Therefore, we can express the temporal complexity or efficiency of the algorithm, directly through C .

In figure 14 (doubly logarithmic) the value of C is showed as a function of the vocabulary size (measured in characters) for the A, D and F threshold degrees. The continuous lines refer to files derived from *APELLIDOS*, and the discontinuous lines refer to files derived from *INTERNA*. In tables 4 and 5 some numerical details are displayed for the threshold degrees A and D.

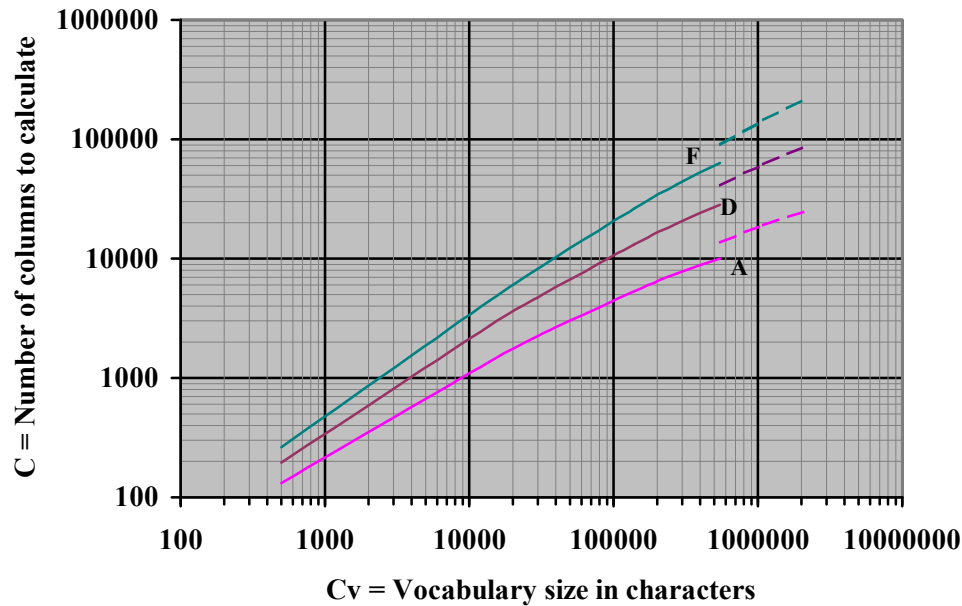


Figure 14: Number of columns to calculate, depending on the vocabulary size

For the most useful threshold degrees (from A to D) and for usual sizes, the percentage of analyzed characters (calculated columns) is between 1% and 10%. That is, only from 0.07 to 0.7 characters for each surname of the vocabulary.

Table 4 : Characters analyzed for the threshold degree A (in %)

File	Characters analyzed	Chars. not analyzed because ...	
		cut-off in trie	common prefix
A-1853	9.94	78.01	12.07
A-3705	7.71	81.77	10.57
A-18523	3.84	89.81	6.48
A-37045	2.67	92.67	4.79
A-73724	1.83	95.17	3.23
I-74391	2.49	92.80	4.86
I-148781	1.73	95.05	3.41
I-297398	1.15	96.84	2.21
U- 31918	3.28	91.56	5.51

Table 5 : Characters analyzed for the threshold degree D (in %)

File	Characters analyzed	Chars. not analyzed because ...	
		cut-off in trie	common prefix
A-1853	19.83	59.76	20.40
A-3705	16.31	64.58	19.11
A-18523	9.49	76.29	14.21
A-37045	7.19	81.46	11.35
A-73724	5.17	86.73	8.09
I-74391	7.50	79.06	13.44
I-148781	5.60	84.19	10.21
I-297398	4.03	88.89	7.08
U- 31918	8.70	78.33	12.97

For Spanish populations between 20,000 and 20,000,000 people, using only the first surname, the following expression allows to estimate the number of characters to be analyzed (the efficiency of the algorithm):

$$C = K * P^{\theta}$$

where K and θ depend on the threshold degree:

for threshold degree A: $K = 145.2$ and $\theta = 0.22$

for threshold degree F: $K = 62.1$ and $\theta = 0.366$

Details about loading and maintaining the trie-tree, can be found in [43].

5. CONCLUSIONS

In the context of APNM, we presented a similarity function, the DEA function, based on the probabilities of the edit operations accordingly to the involved letters and their position, and using a variable threshold. The published works about APNM, that compare several criteria of similarity, produce contradictory results because they use subjective methods for efficacy evaluation. We compared several similarity functions using an evaluation method based on real data and without human relevance judgments. The results of the comparison show that the DEA function has an efficacy significantly higher than the known methods.

In APNM, the similarity criteria more often used are the phonetic codification methods (e.g.: SOUNDEX) because of their good search efficiency although their efficacy is very poor. The distance functions have better efficacy but a poor efficiency. For the DEA function we presented a really efficient search technique, based on a trie-tree structure good enough to be used in a high load IS.

6. REFERENCES

- 1 Newcombe H.B.: *Handbook of record linkage*. Oxford University Press, 1988.
- 2 Barker P. An Analysis of User Input to an X-500 White Pages Directory Service. *IEEE/ACM Trans. on Networking* 1995; **3**(2).

- 3 Veronis J. Computerized correction of phonographic errors. *Computers in Humanities* 1988; **22**: 43-56.
- 4 Berghel H et al. The logic of spelling: Applications of ASM. *PC AI* 1990; **4**(1): 24-27.
- 5 Borgman CL , Siegfried SL. Getty's Synoname and its cousins: A survey of applications of personal name-matching algorithms. *Journal of the American Society for Information Science* 1992; **43**(7): 459-476.
- 6 Levison M. et al. The intelligent detection of second language learner errors. ALLC/ACH 1998.
- 7 Knuth DE. *The art of computer programming. Vol 3 Sorting and Searching*. Ed. Addison Wesley 1975.
- 8 Taft RL. *Name search techniques*. Bureau of Systems Development, New York State Identification and Intelligence System, Albany, NY (Special Rep. No 1) 1970.
- 9 Camps R , Casas R. Codificación fonética de apellidos españoles. *Novática* 1975; **6**.
- 10 Camps R. *Búsqueda por semejanza ortográfica o fonética*. Tesina FIB, Universitat Politècnica de Catalunya, Barcelona, Spain 1981.
- 11 Dart P , Zobel J. Using a pronunciation dictionary for fnetik matching. *CITRI Tech. Report TR-95-28* 1995.
- 12 Pfeifer U. et al. Searching proper names in databases. *Hypertext -Information retrieval-Multimedia. Procee. HIM'95* 1995; 259-275.
- 13 Zobel J , Dart P. Phonetic string matching: Lessons from Information Retrieval. *Proceed.19th Annual Intl. ACM SIGIR Conf. on R&D in IR*. 1996; 166-173.
- 14 Gaizauskas R et al. University of Sheffield: Description of LaSIE System as used for MUC-6. *Proceed. Sixth Message Understanding Conference (MUC-6)*. Morgan Kauffman 1995.
- 15 Attwater DJ , Whittaker SJ. Issues in large-vocabulary interactive speech systems. *BT Technol. Journal* 1996; **14**(1): 177-186.
- 16 French JC et al. Applications of Approximate Word Matching in Information Retrieval. *CIKM'97* 1997; 9-15.
- 17 Davidson L. Retrieval of misspelled names in an airlines passenger record system. *Comm. ACM* 1962; **5**(3): 169-171.
- 18 Fokker DW , Lynch MF. Application of the variety generator approach to searches of personal names in bibliographic databases (Pt I) Microstructure of personal author's names. *Journal of Library Automation* 1974; **7**:105-118.
- 19 Hernansen JC. *Automatic name searching in Large Data Bases of International names*. PhD dissertation, Georgetown University 1985.
- 20 Hall PAV , Dowling GR. Approximate String Matching. *ACM Computing Surveys* 1980; **12**(4): 381-402.
- 21 Kukich K. Techniques for Automatically Correcting Words in Text. *ACM Comp. Surveys* 1992; **24**(4): 377-439.

- 22 Navarro G. A guided tour to approximate string matching. *ACM Computing Surveys* 2001; **33**(1): 32-88
- 23 Bell GB , Sethi A. Matching Records in a National Medical Patient Index. *Comm.ACM* 2001; **44**(9): 83-88.
- 24 Gross A. Getty Synoname: The development of software for personal name pattern matching. *RIAO 91 Conf. Proc. Intelligent text and image handling* 1991; 754-763.
- 25 Siegfried S , Bernstein J. Synoname: The Getty's new approach to pattern matching for personal names. *Computers and the Humanities* 1991; **25**(4): 211-226.
- 26 Pfeifer U. et al. Retrieval effectiveness of proper name search methods. *Information Processing & Management* 1996; **32**(6): 667-679.
- 27 Petrakis EGM , Tzeras K. Similarity Searching in the CORDIS Text Database *Software Practice and Experience* 2000; **13**: 1447-1464.
- 28 Navarro G et al. Matchsimile: A Flexible Approximate Matching Tool for Personal Names Searching. To be published in *JASIST* 2003; **54**(1).
- 29 Gallipi AF. Learning to recognize names across languages. *COLING-96* 1996; 424-429.
- 30 Lopresti D , Wilfong G. Cross-Domain Approximate String Matching. *IEEE String Processing and Information Retrieval Symposium* 1999; 120-127.
- 31 Lowrance R , Wagner RA. An Extension of the String-to-String Correction Problem. *Journ. ACM* 1975;**22**(2): 177-183.
- 32 Oommen BJ. String alignment with substitutions, insertions, deletions, squashing and expansion operations. *Information Science* 1995; **8**(3): 89-107.
- 33 Oommen BJ , Loke RKS. Pattern recognition of strings with substitutions, insertions, deletions and generalized transpositions. *Pattern Recognition* 1997; **30**(5): 789-800.
- 34 Lee J. et al. Efficient algorithms for approximate string matching with swaps. *8th Annual Symposium CPM-97 Proc.* 1997; 28-39.
- 35 Leung VJ. The undecidability of the unrestricted modified edit distance. *Theoretical Computer Science* 1997; **180**(1&2): 203-215.
- 36 Wagner RA , Fischer MJ. The String-to-String Correction Problem. *Journ. ACM* 1974; **21**(1): 168-178.
- 37 Kurtz R. Approximate string searching under weighted edit distance. *Proc.3thd SAWSP* 1996.
- 38 Jouvet D. et al. Speaker-independent spelling recognition over the telephone. *Proc.IEEE Intl.Conf. on Acoustics, Speech and Signal Processing* 1993; 235-238
- 39 Dagan I. et al: Contextual word similarity and estimation from sparse data. *Computer Speech and Language* 1995; **9**: 123-152.
- 40 Weigel A. et al. Lexical post-processing by heuristic search and automatic determination of the edit costs. *ICDAR-95* 1995; 857-860.

- 41 Bunke H , Csirik J. Parametric String Edit Distance and its Application to Pattern Recognition. *IEEE Trans.on Systems, Man, and Cybernetics* 1995; **25**(1): 202-206.
- 42 Ristad ES , Yianilos PN. Learning String-Edit-Distance. *IEEE Trans. on Pattern Analysis and Machine Intelligenc* 1998; **20**(5): 522-532.
- 43 Camps R. *Búsqueda aproximada de antropónimos en las Bases de Datos de los Sistemas de Información en presencia de errores*. PhD dissertation, Departamento LSI, Universitat Politècnica de Catalunya, Barcelona (Spain), 2003.
- 44 Winkler WE. Matching and record-linkage, chap.20 of *Business Survey Methods*. John Wiley & Sons 1995; 355-384.
- 45 Gadd TN: PHONIX: the algorithm. *Program* 1990; **24**(4): 363-366.
- 46 Raghavan et al. Retrieval systems evaluation using recall and precision: Problems and answers. *Proce. of the 12 annual International ACM-SIGIR Conf. on Research and Development in Information Retrieval* 1989; 59-68.
- 47 Saracevic T. Evaluation of Evaluation in Information Retrieval. *ACM/SIGIR'95* 1995.
- 48 Jardine N , Rijsberjen CJ. The use of hierarchic clustering in information retrieval. *Infor. Stor. Retr.* 1971; **7**.
- 49 Rijsbergen CJ. *Information Retrieval*. Ed: Butterworth Scientific Ltd, 2n ed. 1979.
- 50 Navarro G. et al. Indexing Methods for Approximate String Matching. *Bulletin of the IEEECS Tech. Comm. on Data Engineering* 2001; **24**(4): 19-27.
- 51 Fairthorne, R.A.: Empirical hyperbolic distributions. *Journal of Documentation* 1969; **25**(4): 319-343.
- 52 Berghel H , Roach D. An extension of Ukkonen's enhanced dynamic programming ASM algorithm. *ACM Trans. on Inf. Sys.* 1996; **14**(1): 94-106.
- 53 Ukkonen E. Algorithms for approximate string matching. *Information and Control* 1985; **64**: 100-118.
- 54 Gonnet GH , Baeza-Yates R. *Handbook of algorithms and data structures*. Addison-Wesley 2nd ed, 1991; 251-413.
- 55 Myers EW. A sublinear algorithm for approximate keyword searching. *Algorithmica* 1994; **12**: 345-374.
- 56 Oflazer K. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Computational Linguistic* 1996; **22**(1): 73-89.
- 57 Aoe J et al. A Trie Compaction Algorithm for a Large Set of Keys. *IEEE Trans. on Knowledge and Data Eng.* 1996; **8**(3).