A Contextual Normalised Edit Distance

Colin de la Higuera
Laboratoire Hubert Curien
Université de Saint-Etienne
Colin.Delahiguera@univ-st-etienne.fr

Luisa Micó
Dpto. Lenguajes y Sistemas Informáticos
Universidad de Alicante
mico@dlsi.ua.es

Abstract

In order to better fit a variety of pattern recognition problems over strings, using a normalised version of the edit or Levenshtein distance is considered to be an appropriate approach. The goal of normalisation is to take into account the lengths of the strings. We define a new normalisation, contextual, where each edit operation is divided by the length of the string on which the edit operation takes place. We prove that this contextual edit distance is a metric and that it can be computed through an extension of the usual dynamic programming algorithm for the edit distance. We also provide a fast heuristic which nearly always returns the same result and we show over several experiments that the distance obtains good results in classification tasks and has a low intrinsic dimension in comparison with other normalised edit distances.

1 Introduction

In pattern recognition, computational biology and other fields where the data is represented by strings (but also trees and even graphs), having a topology over the intended structures is clearly a reasonable approach. Most useful is the edit or Levenshtein distance that counts the minimum number of modifications needed to get from the first string to the second one [3, 7]. This distance has been thoroughly studied from both a theoretical and a practical point of view [2].

But a number of authors argues that in practice, having to rewrite twice on a string of length 2 is not the same as having to rewrite twice on a string of length 200. For such reasons, several ideas have been proposed to try to relate (in an inverse way) the lengths of the strings with the distance [8].

We propose in this work a new way of normalising the edit distance, by dividing each edit operation *locally* by the length of the string on which it is applied. We argue that this *contextual edit distance* reaches the following compromise:

- It is a metric, and respects the triangle inequality. It can therefore be used for algorithms that rely on this inequality in order not to explore the entire space;
- It corresponds to the nature of normalisations proposed by different authors, as it is closely related with the lengths of the strings [4];
- There exists a heuristic whose computation over-cost is small;
- Preliminary experiments show that the distance discriminates well (and therefore allows an acceleration of fast nearest neighbour approaches), is pertinent, and obtains good classification rates.

After recalling definitions concerning the mathematics and algorithmics of the edit distance (Section 2), we describe in Section 3 this new *contextual* edit distance, prove its validity and provide two algorithms: one that computes exactly the contextual distance and a fast heuristic which reaches the same result in most cases. Experiments are reported in Section 4:

We have compared this measure of distance with the standard edit distance and other normalisations proposed by Marzal and Vidal [4] or by Yujian and Bo [8]. The comparison has been done over three benchmarks and taking into account a variety of parameters: classification rate, speed, capacity of accelerating neighbour search, etc. Finally, in Section 5, we conclude.

2 Definitions

An *alphabet* Σ is a finite nonempty set of symbols. A *string* $x = x_1 \cdots x_n$ is any finite sequence of symbols. We write Σ^* for the set of all the strings over Σ and λ for the empty string. $|x|$ denotes the length of x.

Definition 1 *A distance $d : X^2 \to \mathbb{R}^+$ is a* metric *over X if the following properties hold:*

$$d(x, y) = 0 \iff x = y$$

$$\begin{aligned} d(x,y) &= d(y,x) \\ d(x,y)+d(y,z) &\geq d(x,z) \end{aligned}$$

The advantages of the distance function being a metric are that alternative algorithms and data structures can be used for nearest neighbour algorithms [1]. The triangle inequality can be used to avoid certain computations, resulting in more cost-effective algorithms [6].

2.1 The edit distance

Definition 2 *Given two strings x and y in $\Sigma^\star$, x rewrites into y in one step ($x \rightarrow y$) if one of the following correction rules holds: $x = uav \rightarrow y = uv$ (single symbol deletion); $x = uv \rightarrow y = uav$ (single symbol insertion); $x = uav \rightarrow y = ubv$ (single symbol substitution). We have: $u, v \in \Sigma^\star, a, b \in \Sigma$.*

We will consider the reflexive and transitive closure of this derivation, and $x \xrightarrow{k} y$ if x rewrites into y by k operations of single symbol deletion, single symbol insertion and single symbol substitution. When needed we will write $x \rightarrow_i y$ (respectively $x \rightarrow_s y$ and $x \rightarrow_d y$) to indicate that x rewrites into y trough an insertion (respectively substitution and deletion).

Given 2 strings x and y, the Levenshtein distance between x and y denoted by $d_E(x,y)$ is the smallest k such that $x \xrightarrow{k} y$.

Example 1 *$d_E(abaa, aab) = 2$. $abaa$ rewrites into aab via (for instance) a deletion of the 'b' and a substitution of the last 'a' by a 'b'.*

The well known algorithm [7] for the computation of the edit distance doesn't consider all possible rewriting paths from one string to another (which is infinite), but rather only *internal* paths. To put it simply, each time an insertion is done, the occurrence of the symbol must be in y, each time a deletion takes place, this is of a symbol that was already in x, and each time some symbol a is substituted by a symbol b, then the a must be from x and the b from y. To visualise and study the use of these internal operations, let us mark each symbol appearing in a string by over-lining it ($\overline{a}$) or by underlining it ($\underline{a}$).

Now we consider only internal edit operations of the following sort:

- deleting an over-lined symbol: $u\overline{a}v \rightarrow uv$;
- inserting an underlined symbol: $uv \rightarrow u\underline{a}v$;
- substituting an over-lined symbol $\overline{a}$ by the same symbol, underlined ($\underline{a}$): $u\overline{a}v \rightarrow u\underline{a}v$; This operation has cost 0;
- substituting an over-lined symbol $\overline{a}$ by a different symbol, this time underlined ($\underline{b}$): $u\overline{a}v \rightarrow u\underline{b}v$.

Definition 3 *Let us denote by $d^I_E(x,y)$ the* internal edit distance *between x and y, equal to the distance used by a path starting from $\overline{x}$ and ending in $\underline{y}$ and only using internal edit operations.*

It can easily be checked that: $d_E(x,y) = d^I_E(x,y)$.

Example 2 *$d_E(abaa, baab) \leq 3$ since we have path $abaa \rightarrow bbaa \rightarrow baa \rightarrow baab$ which is internal since it can be marked in the following way:*
$\overline{a}\overline{b}\overline{a}\overline{a} \rightarrow \underline{b}\overline{b}\overline{a}\overline{a} \rightarrow \underline{b}\overline{a}\overline{a} \xrightarrow{0} \underline{b}\underline{a}\overline{a} \xrightarrow{0} \underline{b}\underline{a}\underline{a} \rightarrow \underline{b}\underline{a}\underline{a}\underline{b}$.

If $\pi = (x = w_0 \rightarrow w_1 \rightarrow \cdots \rightarrow w_k = y)$ is an edit path, we will denote by $d_E(\pi)$ the edit weight of the path and by $l_E(\pi)$ the length of the marked path corresponding to it.

Example 3 *Following with Example 2, $l_E(abaa \rightarrow bbaa \rightarrow baa \rightarrow baab)$=5.*

2.2 How can we normalise?

The first ideas that have been proposed and tested by different authors in order to get a distance that might depend on the length of the involved strings have been to divide this distance by some function of the lengths of the strings. We show in this section that the simple ideas used sometimes do not allow the obtained distance to be a metric. Usually the problem is with the triangle inequality.

First, if normalising by dividing by the sum of lengths of the strings $d_{sum}(x,y) = \frac{d_E(x,y)}{|x|+|y|}$ you end up with something that is not a metric: Take $x = ab$, $y = aba$ and $z = ba$ and $d_{sum}(ab, aba) + d_{sum}(aba, ba) = \frac{1}{5} + \frac{1}{5}$ whereas $d_{sum}(ab, ba) = \frac{2}{4}$. Therefore, $d_{sum}(ab, ba) > d_{sum}(ab, aba) + d_{sum}(aba, ba)$ and the triangle inequality no longer holds.

One can also prove that if one takes $d_{max}(x,y) = \frac{d_E(x,y)}{\max(|x|,|y|)}$ or $d_{min}(x,y) = \frac{d_E(x,y)}{\min(|x|,|y|)}$ the distance is still not a metric. In the first case the triangle inequality doesn't hold for $x = ab$, $y = aba$ and $z = ba$ as for d_{max} whereas for d_{min} a counter example can be built with $x = b$, $y = ba$ and $z = aa$.

The well-known normalised edit distance introduced in 1993 by Marzal and Vidal [4] as

$$d_{MV}(x,y) = \min_\pi = \frac{d_E(\pi)}{l_E(\pi)}$$

uses the path with the lowest ratio between the number of edit operations and the length of the path. The authors have shown that this distance is not a metric in the generalised case, but it is still unclear if it is one in the case where the edit costs are 1.

Finally, a normalising process was recently introduced by Yujian and Bo [8]:

$$d_{YB}(x,y) = \frac{2d_E(x,y)}{|x|+|y|+d_E(x,y)}.$$

The authors prove that d_{YB} is a metric. The computation follows simply from the computation of the edit distance. Yet, if we rewrite d_{YB} as:

$$d_{YB}(x,y) = 2 - \frac{2(|x|+|y|)}{|x|+|y|+d_E(x,y)}$$

the influence of the edit distance in the result seems insufficient, specially for very different strings. We give some experimental results in Section 4 which confirm this.

On the other hand, it should be noticed that Yujian and Bo's method (and Marzal and Vidal's) extends to the case where the distance is generalised, *i.e.* where the edit operations have different weights independently of the context [8]. The complexity of their algorithm is quadratic, so if an alternative distance is to do better order it may have to compensate a possible higher computational cost of the individual distances with the fact that less of these need to be computed.

3 The contextual edit distance

The idea is to consider that the weight of **each** edit operation is *context dependent*. If the operation is done in a long string, it will cost less than if it is done in a short string. If $uv \neq \lambda$, and $u \rightarrow v$ (elementary operation), $d_C(u,v) = \frac{1}{max(|u|,|v|)}$.

More precisely, if the operation is a substitution or a deletion the normalised weight is $\frac{1}{|u|}$. If it is an insertion it will be $\frac{1}{|u|+1}$.

Then we can define:

Definition 4 *The* normalised contextual edit distance for path $x = w_0 \rightarrow w_1 \rightarrow \cdots \rightarrow w_k = y$ *is* $\sum_{i=1}^{i=k} d_C(w_{i-1}, w_i)$*. If* π *is the path we will write* $d_C(\pi)$*. The* normalised contextual edit distance between x *and* y *is the minimum value* $d_C(\pi)$ *over all possible paths* π *from* x *to* y*.*

Example 4 *What is* $d_C(ababa, baab)$*? Since we have path* $ababa \rightarrow_d abaa \rightarrow_d baa \rightarrow_i baab$*, the weight of this path is* $\frac{1}{5} + \frac{2}{4} = \frac{7}{10}$*, so* $d_C(ababa, baab) \leq \frac{7}{10}$*. An alternative path is* $ababa \rightarrow_i ababab \rightarrow_d babab \rightarrow_d baab$ *and it follows that* $d_C(ababa, baab)$ *is* $\frac{8}{15}$*.*

3.1 Properties of d_C

Most importantly, we want d_C to respect the conditions from Definition 1: This will allow its use in fast nearest neighbour algorithms.

Theorem 1 d_C *is a metric.*

Proof.

1. $d_C(x,y)$ is well defined. Indeed it can be proved that any path of length more than $|x|+|y|$ has weight more than $\sum_{i=|x|+1}^{i=|x|+|y|} \frac{1}{i} + \sum_{i=|y|+1}^{i=|y|+|x|} \frac{1}{i}$ since the cheapest way is always by using as long intermediate strings as possible. Therefore $d_C(x,y)$ is chosen as the minimum from a finite set and therefore exists.
2. The following are trivial: $\forall x,\, d_C(x,x) = 0$. $d_C(x,y) = 0$ means that no operation has taken place; hence $x = y$. $d_C(x,y) = d_C(y,x)$.
3. The triangle inequality also holds: $\forall\ x,y,z \in \Sigma^\star$ $d_C(x,y) + d_C(y,z) \geq d_C(x,z)$ holds since: (1) If the 'best' path from x to z passes through y, then equality is reached, (2) if not, then (if the triangle inequality didn't hold) we would have a rewriting path $x \xrightarrow{*} y \xrightarrow{*} z$ whose weight would be less than $d_C(x,z)$, which is absurd. ■

We next prove that insertions should be made first.

Lemma 1 *Let* $\Pi_k(x,y)$ *be the set of all paths* π *from* x *to* y *such that* $d_E(\pi) = k$*. Then the shortest path in* $\Pi_k(u,v)$ *for* d_C *is a path* π *where* $\pi = \big(x = w_0 \xrightarrow{n_i}_i w_{n_i} \xrightarrow{n_s}_s w_{n_i+n_s} \xrightarrow{n_d}_d w_{n_i+n_s+n_d} = y\big)$ *with* $n_i + n_s + n_d = k$ *and* n_i *maximal.*

Proof. It is easy to see that the distance is minimised for a given path length by using as many long strings as possible. This is obtained by first making as many insertions as possible and making the substitutions on the longest strings, and finishing with the deletions. ■

We now prove that the contextual edit distance is equal to the internal contextual edit distance, *i.e.* that we only have to consider those paths that correspond to internal operations:

Proposition 1 $d_C(x,y) = d_C^I(x,y)$*.*

Proof. Let us suppose for contradiction that this is not true. Let π be a path from x to y such that $d_C(\pi) = d_C(x,y) < d_C^I(x,y)$. Let us suppose that π is of minimal length between all those paths such that $d_C(\pi) = d_C(x,y)$. We also make use of Lemma 1: The path π is therefore of type $\big(x = w_0 \xrightarrow{n_i}_i w_{n_i} \xrightarrow{n_s}_s w_{n_i+n_s} \xrightarrow{n_d}_d w_{n_i+n_s+n_d} = y\big)$.

We now consider the first edit operation in π which is not internal. There are three cases:

- The first non internal edit operation is a deletion. But since Lemma 1 applies, it cannot be the deletion of a symbol that will be reinserted later. Or it is a symbol that (when marked) is neither over- nor underlined, but that means that another non internal operation occurred earlier.

- The operation is a substitution. Since it is the first non internal operation, it is a substitution of some symbol $\overline{a}$ into a symbol b which will be substituted ($b \rightarrow c$) or erased later ($b \rightarrow \lambda$). Then we can build a cheaper path where the substitution $\overline{a} \rightarrow b$ does not appear at all and where the substitution $b \rightarrow c$ is replaced by $\overline{a} \rightarrow c$, or the deletion $b \rightarrow \lambda$ is replaced by $\overline{a} \rightarrow \lambda$.
- The first non internal operation is the insertion of a symbol that is not underlined. Then this symbol has to be deleted later (or substituted). The path therefore is $x \xrightarrow{*}_i u_1u_2 \rightarrow_i u_1au_2 \xrightarrow{k} v_1av_2 \rightarrow_d v_1v_2 \xrightarrow{*}_d y$. By eliminating both the insertion of a and the deletion of the a, the cost of the new path is $d_C(\pi) - \frac{k+2}{m+1} + \frac{k}{m}$ where $m+1$ is the length of the longest string used in path π whereas k is the number of substitutions in path π (which is clearly at most m). The new path is therefore cheaper than the original one. If the inserted symbol a is in fact substituted by some other symbol b (or $\underline{b}$) then the initial insertion can be replaced by $\lambda \rightarrow b$ (or $\lambda \rightarrow \underline{b}$). ∎

Therefore the contextual edit distance can be computed by

1. computing, for each value k, the maximum number $n_i(k)$ of insertions on a path of length k leading from x to y, and
2. finding the minimum value

$$\sum_{i=|x|+1}^{i=|x|+n_i(k)} \frac{1}{i} + n_s(k) \cdot \frac{1}{|x|+n_i(k)} + \sum_{i=|y|+1}^{i=|y|+n_d(k)} \frac{1}{i}$$

with

- $n_d(k) = |x| - |y| + n_i(k)$, and
- $n_s(k) = k - n_i(k) - n_d(k)$.

3.2 The algorithm

The idea is to use the usual edit distance algorithm. We need, for each pair of prefixes $(x_1 \cdots x_i, y_1 \cdots y_j)$, to compute the maximum number of insertions ($n_i[i][j][k]$) that we can make in a path of length k, where k has to take each possible value of the edit length. Value $-\infty$ for $n_i[i][j][k]$ means that there is no internal path π from $x_1 \cdots x_i$ to $y_1 \cdots y_j$ with $d_E(\pi) = k$. We then deduce n_s. Algorithm 1 is therefore an extension of the usual algorithm to compute the edit distance. Complexity is in $\mathcal{O}\big(|x| \cdot |y| \cdot (|x| + |y|)\big)$. An alternative version making use of quadratic space can easily be deduced by using standard techniques.

Algorithm 1: Computing the contextual distance

```
Data: Two strings $x = x_1 \cdots x_{|x|}$ and $y = y_1 \cdots y_{|y|}$
Result: $B = d_C(x, y)$
for $i : 0 \leq i \leq |x|$ and $j : 0 \leq j \leq |y|$ do
    for $k : 0 \leq k \leq |x| + |y|$ do    $n_i[i][j][k] \leftarrow -\infty$
end
for $i : 1 \leq i \leq |x|$ do $n_i[i][0][i] \leftarrow 0$;
for $j : 0 \leq j \leq |y|$ do $n_i[0][j][j] \leftarrow j$;
for $i : 1 \leq i \leq |x|$ and $j : 1 \leq j \leq |y|$ do
    if $x_i = y_j$ then
        for $k : 0 \leq k \leq |x| + |y|$ do
            $n_i[i][j][k] \leftarrow n_i[i-1][j-1][k]$
    else
        for $k : 1 \leq k \leq |x| + |y|$ do
            $n_i[i][j][k] \leftarrow n_i[i-1][j-1][k-1]$
    for $k : 1 \leq k \leq |x| + |y|$ do
        $n_i[i][j][k] \leftarrow \max\big(n_i[i-1][j][k-1],$
            $n_i[i][j-1][k-1] + 1, n_i[i][j][k]\big)$
end
$B \leftarrow +\infty$;
for $k : 0 \leq k \leq |x| + |y|$ do
    if $n_i[|x|][|y|][k] \geq 0$ then
        $N_i \leftarrow n_i[|x|][|y|][k]$;
        $N_d \leftarrow |x| - |y| + N_i$;
        $N_s \leftarrow k - (N_i + N_d)$;
        $D \leftarrow 0$;
        for $i : |x| + 1 \leq i \leq |x| + N_i$ do
            $D \leftarrow D + \frac{1}{i}$
        $D \leftarrow D + \frac{N_s}{|x| + N_i}$;
        for $i : |y| + 1 \leq i \leq |y| + N_d$ do
            $D \leftarrow D + \frac{1}{i}$
        if $D < B$ then $B \leftarrow D$

end
```

4 Experiments

In order to evaluate the behaviour of the new contextual edit distance we provide experimental results on the performance of the new normalised distance (d_C) by comparing it with other distance functions. We compare our distance with the basic edit distance (d_E), with the normalisation proposed by Yujian and Bo (d_{YB}) [8], and also with the non-metric distance mentioned in Section 2.2, $d_{max}(x,y) = \frac{d_E(x,y)}{\max(|x|,|y|)}$. Finally, comparison with d_{MV}, the distance introduced by Marzal and Vidal [4] is also proposed.

- The intrinsic dimension of the different metric spaces has been analysed through histograms of distances [1];
- We have used a fast nearest neighbour search algorithm to analyse in the new space the average number

of distance computations and search time;

- Using a labelled dataset we have analysed the error rate in a classification task.

To perform these experiments, three datasets were used:

1. A Spanish dictionary with 86062 words (from `http://sisap.org`);
2. A set of 20,660 DNA sequences of genes of Listeria monocytogenes (from `http://sisap.org`);
3. The contour strings of handwritten digits from the NIST SPECIAL DATABASE 3 (from `http://www.nist.gov/srd/nistsd19.htm`).

4.1 Using a heuristic to compute the contextual distance

Experimentally, the complexity of Algorithm 1 is a problem, but it can be shown that the minimum value is obtained very often for $k = d_E(x, y)$. This allows to consider a heuristic called $d_{C,h}$ which is in $\mathcal{O}\big(|x| \cdot |y|\big)$: Instead of computing $n_i[i][j][k]$ for every value of k, only compute it for minimal values of k for which $n_i[i][j][k] \neq -\infty$.

In experiments over the used benchmarks, $d_{C,h}(x, y) = d_C(x, y)$ in 90% of the cases, with differences ranging from 0.03 for the dictionary to 0.008 for the contour strings.

An experiment comparing the histograms of distances between both the contextual distance d_C and the heuristic $d_{C,h}$ was performed. Figure 1 shows that both distances have a very similar behaviour (the intrinsic dimensionality in both cases is similar).

For these reasons, the experiments described in the sequel were run with $d_{C,h}$.

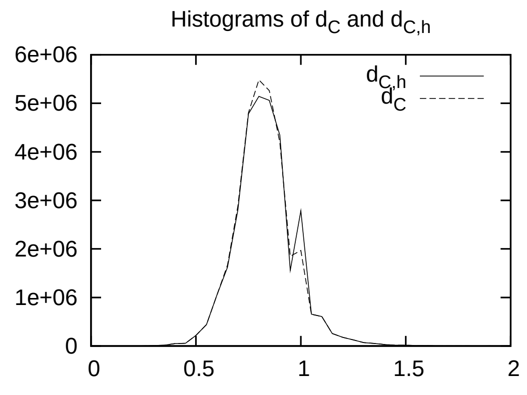


Figure 1. Histograms of distances for the Spanish dictionary using 8000 samples.

4.2 Analysis of the intrinsic dimensionality

Several authors have used histograms of distances to characterise the difficulty of searching in an arbitrary metric space. A quantitative cheap and simple measure of this difficulty was introduced by Chávez *et al* [1]. The *intrinsic dimensionality* of a distance is defined as $\rho = \frac{\mu}{2\sigma^2}$, where μ and σ^2 are the mean and variance of its histograms of distances.

To obtain the intrinsic dimensionality for the datasets, around 1000 strings were used in the case of handwritten digits and genes, and 8000 for the Spanish dictionary.

In Figure 2 are shown the histograms of distances for the *genes* dataset with the four normalised distances (top), and the histogram of distances for the edit distance (bottom). The same experiments have been performed with the Spanish dictionary dataset and the handwritten digits dataset (for lack of space, histograms for the Spanish dictionary and handwritten digits dataset are not shown).

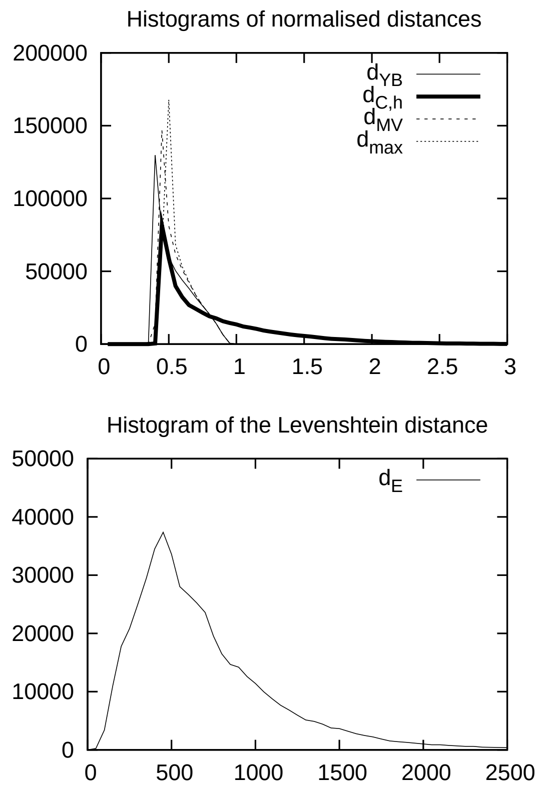


Figure 2. Histograms of distances for genes.

In the three datasets, the histogram for the other normalised edit distances appear to be more concentrated than the contextual and Levenshtein distances. Moreover, this result is consistent with the computation of the intrinsic dimension, which is shown in Table 1.

Table 1. Intrinsic dimensionality

	Datasets		
Distances	Spanish D.	hand. digits	genes
d_{YB}	40.57	18.81	8.43
$d_{C,h}$	18.61	7.95	1.88
d_{MV}	33.98	19.36	11.25
d_{max}	30.25	19.48	14.13
d_E	8.75	4.91	0.99

We believe that the contextual edit distance reflects better the fact that by computing the cost of each edit operation in the moment the operation takes place, strings of different length are going to be further apart (different) than with more global methods. This is usually considered to be favourable to the distance which thereby discriminates more.

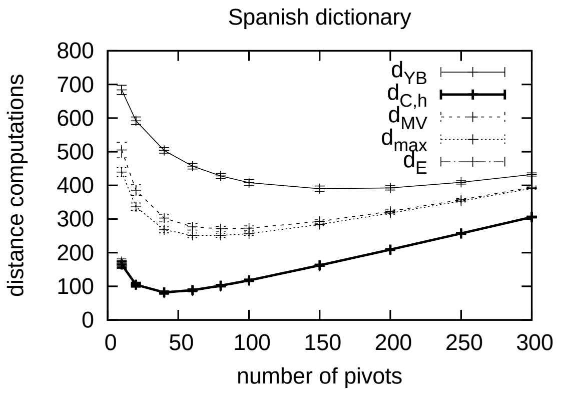
4.3 Experiments with fast nearest neighbour search algorithms

When the intrinsic dimensionality of the space increases, the performance of fast nearest search algorithms decreases dramatically. Many fast nearest neighbour methods are based on the use of the triangle inequality property. When the histograms of distances are concentrated, the distance between two random distances are closer to zero and the probability of discarding an element during search is lower [1].

To analyse how the intrinsic dimensionality is related with the difficulty of searching in a metric space, in the following experiments we used the nearest neighbour search algorithm LAESA [5]. This algorithm is based on the use of the triangle inequality to speed up the search of the nearest neighbour. The main characteristics of the algorithm are that it only requires linear preprocessing time and memory to find the nearest neighbour, with an average constant number of distance computations. In preprocessing time, the distance between the dataset and a selected subset of elements (pivots) is stored to use during the search. In the literature there exist other methods that also use the metric properties of the distances to accelerate the search, and we argue that our results will apply in similar cases.

To carry out the experiments of this section, test data for the Spanish dictionary have been generated using the program `genqueries` (can be found in the Metric Spaces Library from the SISAP Web site) with a perturbation of two operations over the training dataset.

All the experiments were repeated with ten different prototype sets (with 1000 training samples) and 1000 different samples. Therefore, results were obtained as an average over 10000 experiments. Deviations are shown in the figures. Results are shown in Figures 3 and 4. In these experiments, only the average numbers of distance computations and of the search times have been evaluated (all the experiments have been performed with Intel Core 2 Duo processor, 2.4GHz with 1 GB RAM). It is interesting to see that the average number of distance computations by the Levenshtein and the contextual distance are similar with two very different datasets (handwritten digits and Spanish dictionary). The computation time of the contextual distance is around twice the computation time of the Levenshtein distance, but this is compensated by a largely inferior number of times the distance has effectively to be computed.



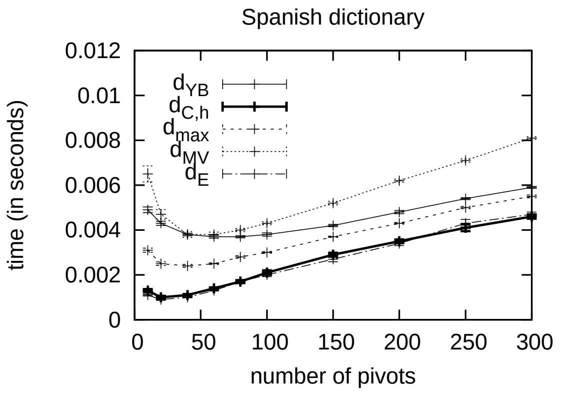
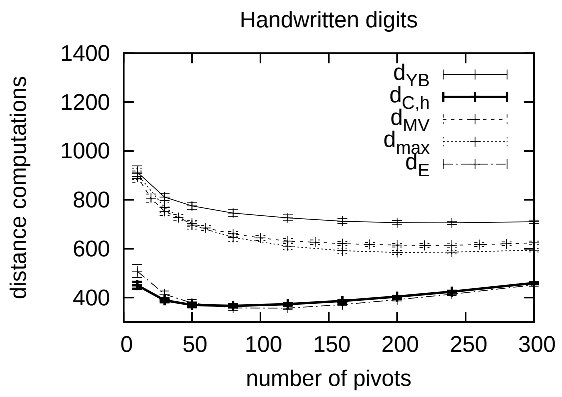


Figure 3. Average number of distance computations and search times using different numbers of pivots in the fast algorithm for the Spanish dictionary.



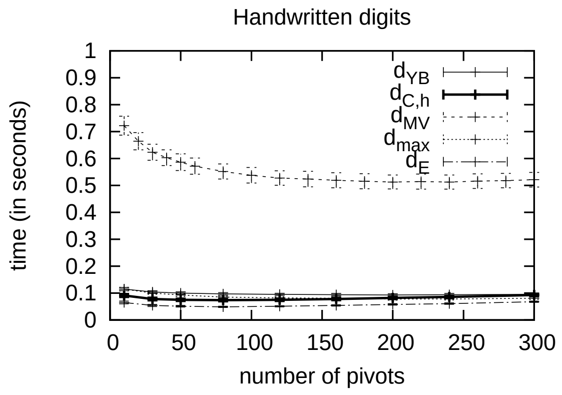


Figure 4. Average number of distance computations and search times using different numbers of pivots in the fast algorithm for the handwritten digits.

4.4 Application to a classification task

If the objects belonging to the training data are labelled, we can use nearest neighbour search for classification: When a new unlabelled test sample is used as a query, this object is classified with the same label as its nearest neighbour in the training set. In the case where the true label of the sample and the predicted label don't match, this is counted as a mistake.

As the handwritten digits dataset is labelled, we have used this data. There has been no preprocessing of the digits: Orientation and sizes are therefore widely different from scribe to scribe, as can be seen in Figure 5.

For this experiment, around 1000 digits (100 by class) have been used as training, and a further 1000 digits (from different writers) have been used as test. All the experiments were repeated with ten different prototype sets and 1000 different samples. Therefore, results are given as an average over 10000 experiments.

The results of the classification task are shown in Table 2. As expected, all the normalisations (metric or

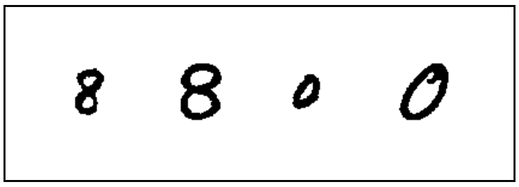


Figure 5. Different '8' and '0' from the NIST database

not) obtain higher success rates than when normalisation is not applied. It should be noted that the normalisation with best results is d_{max}, but one should remember that, as shown in Section 2.2, it is not a metric. Additional experiments show that the same error rate is obtained when the exact contextual distance algorithm is used instead of the heuristic.

Table 2. Error rate using different distances in a handwritten digits classification task.

	handwritten digits	
Distances	LAESA	Exhaustive search
d_{YB}	5.19	5.22
d_{MV}	5.04	5.04
d_C	5.30	5.30
$d_{C,h}$	5.30	5.30
d_{max}	4.85	4.86
d_E	6.19	6.26

5 Conclusions and further works

To summarise, we have proposed a new extension of the edit distance with the following properties:

- the contextual edit distance is able to take into account the length of the strings;
- the contextual edit distance can be computed in cubic time;
- the contextual edit distance is a metric.

The intrinsic dimension associated with this contextual distance is low. On the other hand classification rates are good and the nature of the distance allow to gain in techniques where nearest neighbours are searched.

In future works, an adaptation of the technique to the generalised edit distance should be considered. The naive idea of using these proposals directly fails: It is easy to build an example where the best path is reached by inserting some symbol cheaply in order to have a long string in which the necessary (expensive) substitutions take place; Then the dummy symbols are erased.

An even more important open question concerns the cubic complexity of Algorithm 1; This is clearly too high, and further analysis is needed in order to better it.

6 Acknowledgements

The authors thank the Spanish CICyT for partial support of this work through projects DPI2006-15542-C04-01, the IST Programme of the European Community, under the PASCAL Network of Excellence, IST–2002-506778, the program CONSOLIDER INGENIO 2010 (CSD2007-00018), and the ANR (for program BLAN07-1_184534).

References

[1] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[2] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[3] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965.

[4] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):926–932, 1993.

[5] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.

[6] J. R. Rico-Juan and L. Micó. Comparison of AESA and LAESA search algorithms using string and tree-edit-distances. *Pattern Recognition Letters*, 24(9-10):1417–1426, 2003.

[7] R. Wagner and M. Fisher. The string-to-string correction problem. *Journal of the ACM*, 21:168–178, 1974.

[8] L. Yujian and L. Bo. A normalized Levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1091–1095, 2007.