Abstract—The Normalized Edit Distance (NED) between two strings $X$ and $Y$ is defined as the minimum quotient between the sum of weights of the edit operations required to transform $X$ into $Y$ and the length of the editing path corresponding to those operation strings. An algorithm for computing the NED has recently been introduced by Marzal and Vidal that exhibits $O(mn^3)$ computing complexity, where $m$ and $n$ are the lengths of $X$ and $Y$. We propose here an algorithm that is observed to require in practice the same $O(mn)$ computing resources as the conventional unnormalized Edit Distance algorithm does. The performance of this algorithm is illustrated through computational experiments with synthetic data, as well as with real data consisting of OCR chain-coded strings.

Index Terms—Normalized edit distance, Levenshtein distance, pattern recognition, string correction, editing, spelling correction, optical character recognition, speech recognition, fractional programming, fast algorithms.

I. INTRODUCTION

The Normalized Edit Distance (NED) between strings $X$ and $Y$, $d(X, Y)$, is defined as the minimum of $W(P)/L(P)$, where $P$ is an editing path between $X$ and $Y$, $W(P)$ is the sum of the weights of the elementary edit operations in $P$ and $L(P)$ is the number of these operations (Length of $P$). As was shown in [5], $d(X, Y)$ cannot be obtained by “post-normalization”; that is, first computing the conventional (unnormalized) edit distance between $X$ and $Y$ (i.e., minimum of $W(P)$) and then normalizing this distance by the length of the corresponding editing path. In order to correctly compute NEDs, an algorithm was introduced in [5] which obtains $d(X, Y)$ with $O(mn^3)$ computing complexity, where $m$ and $n$ are the lengths of $X$ and $Y$ and $m \geq n$. The usefulness of NEDs was also illustrated in [5] through hand-written digit recognition experiments based on the $k$-Nearest-Neighbor classification technique, in which NED consistently outperformed both the unnormalized and postnormalized edit distances. However, these unnormalized or “suboptimally normalized” edit distances (and many other variations of the same), can be computed in $O(mn)$ time. Clearly, in some practical situations such a lesser computational complexity can outweigh the benefits of the optimality of NED.

In this paper, an algorithm is introduced which is observed to obtain the correct NED with almost the same $O(mn)$ asymptotical computational complexity as the conventional (suboptimal or unnormalized) techniques do. More specifically, this algorithm obtains the NED by repeatedly computing a number of conventional edit distances. This number is generally very small and is observed not to significantly depend on the length of the compared strings. This algorithm is based on a technique known as “Fractional Programming.”

II. FRACTIONAL PROGRAMMING

Fractional Programming (FP) [2], [11] is an optimization technique that can be useful in many problems involving ratio functions. It can be considered as a particular case of the so-called “C-programming” [11] which can further deal with more
general families of functions. FP seeks to solve the following family of problems:\footnote{We consider here minimization problems rather than maximization problems as in [11]. It can be easily verified that the same theorems, proofs and the algorithm of [11] also hold in our formulation.}

**Problem Q.** Find

$$q^* = \min_{z \in \mathbb{Z}} \frac{u(z)}{v(z)}$$

where $u, v : \mathbb{Z} \rightarrow R; v(z) > 0, \forall z \in \mathbb{Z}$.

The set of optimal solutions to $\text{Q}$ is denoted as $Z^*$; i.e.,

$$Z^* = \{ z \in \mathbb{Z} : \frac{u(z)}{v(z)} = q^* \}.$$  

The parametric method of FP allows us to solve $\text{Q}$ if a solution is available for a parametric problem of the type:

**Problem Q(\lambda).** Find

$$q^*(\lambda) = \min_{z \in \mathbb{Z}} \left( u(z) - \lambda v(z) \right)$$

where $\lambda \in \mathbb{R}; u, v : \mathbb{Z} \rightarrow R; v(z) > 0, \forall z \in \mathbb{Z}$.

The set of optimal solutions to $\text{Q}(\lambda)$ is denoted as $Z^*(\lambda)$. The following theorem establishes that, in fact, a $A^*$ solution of this algorithm can be easily established as shown in Fig. 1, searches for such a solution of $q^*(\lambda) = 0$ (and a corresponding $z^* \in Z^*$). The correctness of this algorithm can be easily established as follows [11]:

**Algorithm Dinkelbach**

$$z^* := \text{arbitrary element}(\mathbb{Z})$$

$$\lambda^* := \frac{u(z^*)}{v(z^*)}$$

repeat

$$\lambda' := \min_{z \in \mathbb{Z}} \frac{u(z)}{v(z)} - \lambda^* v(z)$$

$$\lambda^* := \frac{u(z^*)}{v(z^*)}$$

until $\lambda^* = \lambda'$

return ($z^*, \lambda^*$)

end Dinkelbach

**Theorem 1.** \footnote{We consider here minimization problems rather than maximization problems as in [11]. It can be easily verified that the same theorems, proofs and the algorithm of [11] also hold in our formulation.}

1) $z \in Z^*$ if $z \in Z^*(u(z)/v(z));$

2) the equation $q^*(\lambda) = 0$ has $\lambda^* = q^*$ as its unique solution.

Dinkelbach's algorithm [2], shown in Fig. 1, searches for such a solution of $q^*(\lambda) = 0$ (and a corresponding $z^* \in Z^*$). The correctness of this algorithm can be easily established as follows [11]:

**Algorithm FPNED**

$$z^* := \text{arbitrary.element}(\mathbb{Z})$$

$$\lambda^* := \frac{u(z^*)}{v(z^*)}$$

repeat

$$\lambda' := \min_{z \in \mathbb{Z}} \frac{u(z)}{v(z)} - \lambda^* v(z)$$

$$\lambda^* := \frac{u(z^*)}{v(z^*)}$$

until $\lambda^* = \lambda'$

return ($z^*, \lambda^*$)

end FPNED

**Theorem 2.** If $Z$ is finite, Dinkelbach's algorithm terminates with $\lambda^* = q^*$ and $z^* \in Z^*$; otherwise, the sequence of values of the variable $\lambda^*$ that it generates converges superlinearly to $q^*$.

III. FRACTIONAL PROGRAMMING AND NORMALIZED EDIT DISTANCE: INITIALIZATION PROCEDURES

Let $\Sigma$ be an alphabet and let $e$ be the symbol for the empty string. Let $(a \rightarrow b)$ be an elementary edit operation, where $a$ and $b$ are strings of length 0 or 1 and $(a \rightarrow b) \neq (e \rightarrow e)$. Each elementary edit operation $(a \rightarrow b)$ is assumed to be weighted by a nonnegative weight function $\chi(a \rightarrow b) \in \mathbb{R}^+$.

Let $X, Y \in \Sigma^*$ be two strings over $\Sigma$. The string $X$ can be transformed into $Y$ through a certain sequence of edit operations, which can be seen as an "editing path" between $X$ and $Y$. For each $P = (i_0, j_0), \ldots, (i_l, j_l)$, let $\mathcal{L}(P) = m$ be the length of $P$ and let $W(P) = \sum_{i=1}^{m} Y_{j_{i+1}+\ldots+j_l}$ be the weight of $P$. The computation of the NED between $X$ and $Y$ can then be formally stated as the following minimization problem [5]:

**Problem NED:** Find

$$d^* = d(X, Y) = \min_{P \in \mathcal{P}} \frac{W(P)}{\mathcal{L}(P)}.$$
INITIALIZATION. With the conventional unnormalized Edit-Distance path (Fig. 3b):

\[ P^* = (0,0), (1,1), (2,2), (3,3), (4,4) \]

\[ \lambda^* = \frac{W(\lambda^*)}{L(\lambda^*)} = \frac{3}{2} \]

FIRST ITERATION. \( P^* = \text{argmin}_{P} (W(P) - 3/2 L(P)) \) is solved with the unnormalized Edit-Distance algorithm, yielding (Fig. 3c):

\[ P^* = (0,0), (1,1), (2,2), (3,3), (4,4) \]

\[ \lambda^* = \frac{3}{2} = \frac{3}{2} \]

This is already a solution to NED [5] but, since \( \lambda^* = 4/3 \neq 3/2 \) = \( \lambda' \), an additional iteration is required to guarantee that this is in fact a correct result.

SECOND AND LAST ITERATION. \( P^* = \text{argmin}_{P} (W(P) - 4/3 L(P)) \)

is solved, yielding:

\[ P^* = (0,0), (1,1), (2,2), (3,3), (4,4) \]

\[ \lambda^* = \frac{4}{3} = \frac{4}{3} \]

Although the conventional unnormalized Edit Distance constitutes an adequate initialization procedure for FPNED computation, better convergence behavior can be achieved by using a suboptimal NED computing technique known as "Locally Normalized Edit Distance" (LNED). This heuristic consists of locally minimizing, at each point of the computational lattice, the quotient of the current path weight by the current path length. It is easy to show by means of counter examples that, in general, this technique fails to obtain the true NED. However, this approach has been proposed in the field of Automatic Speech Recognition [1], [4] as an (empirically better) alternative to the conventional Dynamic Time Warping procedure usually adopted for comparing acoustic sequences of speech [10] [8]. As adapted to our NED problem, this suboptimal approach can be implemented as shown in Fig. 4.

Algorithm LNED

```
var W = array [0..|X|, 0..|Y|] of R
L = array [0..|X|, 0..|Y|] of N
i, j, l, m \in N; W, L \in R

W_{0,0} := 0; L_{0,0} := 0
for i := 1 to |X| do W_{i,0} := W_{i-1,0} + \gamma(X_i \rightarrow \epsilon); L_{i,0} := L_{i-1,0} + 1 endfor
for j := 1 to |Y| do W_{0,j} := W_{0,j-1} + \gamma(\epsilon \rightarrow Y_j); L_{0,j} := L_{0,j-1} + 1 endfor
for i := 1 to |X| do
  for j := 1 to |Y| do
    W_{i,j} := W_{i-1,j} + \gamma(X_i \rightarrow Y_j); L_{i,j} := L_{i-1,j} + 1
    W' := W_{i,j} + \gamma(\epsilon \rightarrow \epsilon); L' := L_{i,j} + 1
    if W' > L' then W_{i,j} := W'; L_{i,j} := L' endif
    W' := W_{i,j} + \gamma(\epsilon \rightarrow Y_j); L' := L_{i,j} + 1
    if W' > L' then W_{i,j} := W'; L_{i,j} := L' endif
  endfor
endfor

return (W_{|X|,|Y|}, L_{|X|,|Y|})
end LNED
```

Fig. 4. Locally Normalized Edit Distance algorithm.

It can be easily seen that the computational complexity of this heuristic is essentially the same as that of the conventional unnormalized Edit Distance and, as will be seen later on, the results are often closer to the optimal NED, so that it is clearly a better candidate for initializing the FPNED algorithm.

IV. EXPERIMENTS

In order to test the performance of the FPNED algorithm in practice, two computational experiments were carried out. The first experiment dealt with synthetic-data. It aimed at establishing comparisons between the FPNED computing complexity growth and the corresponding growth of both the conventional unnormalized Edit-Distance algorithm (ED) [12] and the Dynamic Programming procedure that we had previously introduced for computing NEDs (DPNED algorithm) [5]. In the second experiment we adopted the same real data set used in the Edit-Distance-based hand-printed digit recognition experiments presented in [5], and compared the computing performance of the proposed FPNED algorithm with that of the basic algorithm.

For the first experiment, strings over an alphabet \( \Sigma \) of 16 symbols were randomly generated, with lengths running from 2 up to 1,024 in powers-of-2 increments. For each length, 10 strings were generated and different algorithms were applied for computing both the conventional Edit Distance and the Normalized Edit Distance between all 100 pairs of strings of this length. A single (asymmetrical) \( \gamma \) function was also randomly generated for the whole experiment, with real values in the range \([0, 1]\) and with same-symbol substituting weight \( \gamma(a \rightarrow a) = 0 \), \( \forall a \in \Sigma \). The results are presented in Fig. 5 which shows the computation performance of all the algorithms with respect to that of the basic conventional unnormalized Edit Distance algorithm (ED). For the initialization of the FPNED algorithm, both the ED and the LNED algorithms were considered. Computation performance has been measured in terms of number of "core" (unit-cost) computing operations required by the different algorithms, each core operation consisting of a local minimization among the three basic alternatives (insertion, deletion, substitution) and the corresponding arithmetic operations involved.

![Fig. 5. Performance of different algorithms for the computation of the Normalized Edit Distance, with respect to the cost of computing the conventional (unnormalized) Edit Distance (ED). DPNED is the Dynamic Programming algorithm of [5]; and FPNED is the Fractional Programming algorithm here proposed, initialized with ED and with the Locally Normalized Edit Distance (LNED).](image-url)
It is worth noting that the FPNE D technique exhibits an average computing complexity that grows almost at the same rate as that of the basic ED algorithm does. The ratio between the computing time of FPNE D and that of ED is almost a constant factor (ranging from 2 to 2.65 for Locally Normalized Edit Distance (LNED) initialization and from 2.5 to 3.5 for ED initialization). In contrast, the computing performance of the previous Dynamic Programming Normalized Edit Distance (DPNE D) algorithm [5] grew far faster than both the basic ED and both versions of FPNE D \( O(n^2) \) versus \( O(n^r) \) when the actual editing path is required, or \( O(n) \) versus \( O(n^r) \) if only the distance is needed.

In the second experiment, the data consisted of 500 chain-coded strings representing hand-written digits (OCR), with an (asymmetrical) weight function obtained from the probabilities of insertion, deletion and substitution errors for the different chain-codes, as supplied by the ECOI learning algorithm [9] [5]. The computing performance in this task, averaged over all 250,000 pairs of strings, is presented in Table I for different algorithms. It is worth noting that the FPNE D algorithm can obtain correct results by just computing the basic (LN)ED algorithm 2.03 times on the average. In contrast, the previous DPNE D procedure is more than one order of magnitude less efficient.

![Table I](image)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Average number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ED (or LNED)</td>
<td>1.00</td>
</tr>
<tr>
<td>FPNE D-Init = LNED</td>
<td>2.03</td>
</tr>
<tr>
<td>FPNE D-Init = ED</td>
<td>2.96</td>
</tr>
</tbody>
</table>

Although not observed in any of the experiments, it should be noted that the worst-case computing cost of the FPNE D algorithm can, in theory, become much worse than the average figures reported. This is due to the iterative nature of the algorithm: Although finite convergence is actually guaranteed, it is difficult to find a (theoretical) bound for the number of iterations in realistic situations.

V. CONCLUDING REMARKS

From the experiments presented in the last section, it is clear that correct computation of Normalized Edit Distances is no longer a problem in practice. Only one possible difficulty may remain in the case that computation needs to be performed on-line with one of the strings as, e.g., in certain real-time applications. In this case, optimal results can only be obtained with our previous cubic complexity algorithm [5] and further research would be required to develop fast on-line computation techniques for NEDs.

The use of correct NEDs, rather than the conventional unnormalized Edit Distance and/or heuristic or suboptimal versions of NED, is thought to lead to improvements in practically all fields in which Edit Distances are used to compare objects. Some experiments with OCR data (hand-written digits) clearly supporting this assertion were presented in [5]. But many other applications become apparent.

A particularly interesting case is the procedure usually adopted in Automatic Speech Recognition to assess the performance of continuous Speech recognizers. In this case, error rates are measured in terms of the "relative" minimum number of words (or phonemes) that have been substituted, inserted or deleted by the recognizer with respect to the reference (correct) transcription of the test utterances [6]. The word "relative" is used to express a normalization by the number of words in the reference transcription. Obviously, this is not the only possible normalization criterion and many others can be (and have in fact been) adopted. This has been studied in detail in [7], with the conclusion that the use of a criterion closely related with NED exhibits many desirable properties.

As a conclusion to this paper, we would like to remark that Fractional Programming (and the more general C-Programming as well) [11] constitute a very important computational tool that allows us to extend known solutions to certain problems to more general and interesting settings for these problems. Apart from the development that we have described here, another example worth mentioning is an optimization problem in the context of stochastic (HMM) modeling [3] which has recently been solved using an iterative technique that can be considered as closely related to FP. By looking at Fractional (or C-) Programming from its most general perspective, we think that interesting improvements can be easily found to many other problems of Pattern Recognition for which good, but not perfect, solutions are already available.

ACKNOWLEDGMENTS

This work was partially supported by the Spanish CICYT under Grant No. TIC 1026/92–C02. Andrés Marzal’s work was carried out while he was with Departamento de Sistemas Informáticos y Comunicación, Universidad Politécnica de Valencia.

REFERENCES