Efficient Algorithms For Normalized Edit Distance

ABDULLAH N. ARSLAN[1], *Department of Computer Science, University of California, Santa Barbara, CA 93106, USA. E-mail: arslan@cs.ucsb.edu*

ÖMER EGECIOGLU[2], *Department of Computer Science, University of California, Santa Barbara, CA 93106, USA. E-mail: omer@cs.ucsb.edu*

ABSTRACT: A common model for computing the similarity of two strings X and Y of lengths m and n respectively, with $m \geq n$, is to transform X into Y through a sequence of edit operations, called an *edit sequence*. The edit operations are of three types: insertion, deletion, and substitution. A given cost function assigns a weight to each edit operation. The *amortized weight* for a given edit sequence is the ratio of its weight to its length, and the minimum of this ratio over all edit sequences is the *normalized edit distance*. Existing algorithms for normalized edit distance computation with proven complexity bounds require $O(mn^2)$ time in the worst-case. We give provably better algorithms: an $O(mn \log n)$-time algorithm when the cost function is *uniform*, i.e, the weights of edit operations depend only on the type but not on the individual symbols involved, and an $O(mn \log m)$-time algorithm when the weights are rational.

Keywords: Edit distance, normalized edit distance, algorithm, dynamic programming, fractional programming, ratio minimization.

1 Introduction

Measuring similarity of strings is a basic problem in computer science with applications in many fields such as computational biology, text processing, optical character recognition, image and signal processing, error correction, information retrieval, pattern recognition, and pattern matching in large databases.

Given two strings X and Y over a finite alphabet whose lengths are m and n respectively with $m \geq n$, we consider sequences of weighted *edit operations* (insertions, deletions, and substitutions of characters), by means of which X is transformed into Y. If we call each such sequence an *edit sequence*, then the ordinary (conventional) *edit distance problem* (ED) seeks for an edit sequence with minimum total weight over all edit sequences. The *edit distance* between X and Y is defined as the weight of such a sequence. Although the edit distance is a useful measure for similarity of two strings, for some applications the lengths of the strings compared need to be taken

[1]Supported in part by a UCSB-COR research grant 1999–2000.
[2]Supported in part by NSF Grant No. CCR–9821038.

into account. Two binary strings of length 1000 differing in 1 bit may have the same edit distance as two binary strings of length 2 differing in 1 bit, although one would most likely state that only the 1000-bit strings are “almost equal”. Marzal and Vidal considered an alternate measure called *normalized edit distance* (NED) between X and Y [7] that takes the lengths into account. If we define the amortized weight of an edit sequence as the ratio of the total weight of the sequence to the length of the sequence, NED is the minimum amortized weight over all edit sequences. Marzal and Vidal showed that NED yields better results in empirical experiments. However, it seems that the computation of NED requires significantly more work than those of ordinary edit distance algorithms, which are mostly dynamic programming based. It is also interesting to note that NED cannot be obtained by “post-normalization”, i.e., first computing the ordinary edit distance and then dividing it by the length of the corresponding edit sequence [7].

Ordinary edit distance can be computed in $O(mn)$ time [13, 17, 5], or $O(mn/\log n)$ time if the weights are rational [8]. In order for NED computations to be advantageous, the computational complexity of an algorithm for the latter should not significantly exceed these.

There are several algorithms to compute NED, both sequential [7, 11, 16] and parallel [4]. Observing that the length of an edit sequence lies in the range m and $m+n$ inclusive, an $O(mn^2)$-time dynamic programming algorithm can be developed for this problem [7]. Furthermore, it has been noted that NED can be formulated as a special case of *constrained edit distance* problems [11]. By adapting the techniques used for the constrained edit distance problems, NED can be computed in $O(smn)$ time where s is the number of substitutions in an optimal edit sequence [11]. But since s can be as large as n, the worst case time complexity remains $O(mn^2)$.

Another approach for NED computation uses *fractional programming* [16]; an iterative method in which an ordinary edit distance algorithm is invoked at each iteration. Experimental results on both randomly generated synthetic data, and real applications performed by Vidal, Marzal, and Aibar [16] suggest that the number of iterations necessary for the NED computation with this method is bounded by a small constant. This implies an achievement of experimental $O(mn)$ time complexity for NED computation, but as argued by Vidal, Marzal, and Aibar, a mathematical proof of a theoretical bound for this algorithm seems difficult, although it may be possible to analyze at least the average time complexity under a reasonable probabilistic model.

In this paper, we direct our attention to NED computations for common types of cost functions. First, we analyze the case when the cost function is *uniform*; i.e. weight of an edit operation does not depend on the symbols involved in the operation, but only on its type. In this case, we classify the edit operations into four types : insertion, deletion, matching and non-matching substitutions. Second, we analyze the case when the weights are *rational*, i.e. they are integral multiples of a fixed real number, but not necessarily uniform. These and more restricted cases of cost functions have been studied in the literature in the context of ordinary edit distances [12, 15, 10, 8].

The previously suggested algorithms for NED do not specialize to faster than $O(mn^2)$-time algorithms for the cases we analyze in this paper. We propose provably better algorithms for NED computation in these cases. Our algorithms are similar to the fractional programming normalized edit distance algorithm in that they iteratively solve certain subproblems that involve shortest path computations. Each of these subproblems is solved by a standard ordinary edit distance algorithm, such as the one proposed by Wagner and Fisher [17]. But unlike the fractional programming algorithm, our algorithms perform a binary search for the optimum value of NED by making use of a technique developed by Megiddo [9] for the optimization of the ratio of two linear functions. The worst-case resulting time complexity of our algorithms are $O(mn \log n)$ when the weights are uniform (Corollary 4.4), and $O(mn \log m)$ when the weights are rational (Corollary 4.9).

The outline of this paper is as follows. Section 2 consists of preliminaries; definitions, notation used, and the problem statement. In section 3 we describe the optimization of the ratio of two linear functions and Megiddo's method. Section 4 outlines our algorithms and gives proofs of their worst-case time complexities. This is followed by remarks in section 5 and conclusions in section 6.

2 Definitions

Let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$ be two strings over an alphabet Σ with $m \geq n \geq 0$, not both null. We assume that the edit operations applicable on the symbols of X to transform it into Y are of three types: inserting a symbol into X, deleting a symbol from X, or substituting a symbol from Σ for a symbol of X. The substitution operation can further be broken down into matching and non-matching substitutions.

More formally for $1 \leq i \leq m$, the allowable edit operations are

(1) *Insertion*: any symbol $s \in \Sigma$ can be inserted before or after x_i,

(2) *Deletion*: the symbol x_i can be deleted.

(3) *Substitution*: the symbol x_i can be replaced by a symbol $s \in \Sigma$.

A substitution operation is

(3-a) a *matching* substitution if $s = x_i$,

(3-b) a *non-matching* substitution if $s \neq x_i$.

A common framework for edit distances is the *edit graph* $G_{X,Y,\gamma}$ of the strings X and Y, and a given cost function γ . The edit graph is a weighted directed acyclic graph having $(m+1)(n+1)$ lattice points (i, j) for $0 \leq i \leq m$, and $0 \leq j \leq n$ as vertices. The cost function γ determines the edge weights as we will explain later. The top-left extreme point of this rectangular grid is labeled $(0, 0)$ and the bottom-right extreme point is labeled (m, n), as shown in figure 1. The arcs of $G_{X,Y,\gamma}$ are divided into three types corresponding to edit operations:

(1) *Horizontal arcs*: $\{((i-1, j), (i, j)) \mid 0 < i \leq m, 0 \leq j \leq n\}$ (deletions).

(2) *Vertical arcs*: $\{((i, j-1), (i, j)) \mid 0 \leq i \leq m, 0 < j \leq n\}$ (insertions).

(3) *Diagonal arcs*: $\{((i-1, j-1), (i, j)) \mid 0 < i \leq m, 0 < j \leq n\}$ (substitutions).

If $x_i = y_j$, then the diagonal arc $((i-1, j-1), (i, j))$ is a *matching diagonal arc*, otherwise a *non-matching diagonal arc*. Figure 1 illustrates an example edit graph for $X = aba$ and $Y = bab$.

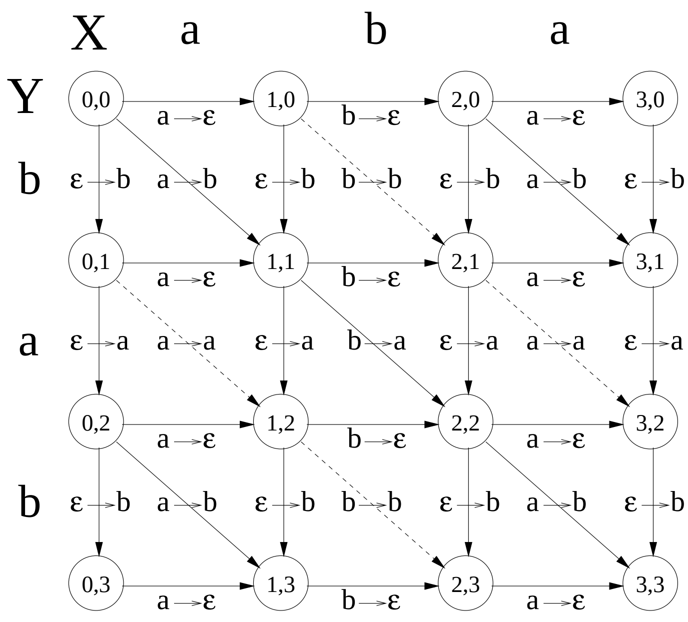

FIG. 1: The edit graph $G_{X,Y,\gamma}$ for the strings $X = aba$ and $Y = bab$. The cost function γ is not specified in this case.

An *edit path* in $G_{X,Y,\gamma}$ is a directed path from $(0,0)$ to (m,n). Steps of an edit path correspond to an edit sequence as follows: A horizontal arc $((i-1,j),(i,j))$ corresponds to the deletion of x_i (i.e. $x_i \rightarrow \epsilon$), a vertical arc $((i,j-1),(i,j))$ corresponds to the insertion of y_j immediately before x_i (i.e. $\epsilon \rightarrow y_j$), and a diagonal arc $((i-1,j-1),(i,j))$ corresponds to the substitution of symbol y_j for x_i (i.e. $x_i \rightarrow y_j$), where ϵ represents the null string. In figure 1, the arcs are labeled accordingly.

A cost function γ assigns a weight to each edit operation, turning $G_{X,Y,\gamma}$ into a weighted graph. γ is given as a cost matrix where rows and columns are indexed by the symbols in $\{\epsilon\} \cup \Sigma$. The entry at row s_1, and column s_2 is the weight $\gamma_{s_1 \rightarrow s_2}$ of the edit operation $s_1 \rightarrow s_2$. Note that $\gamma_{\epsilon \rightarrow \epsilon}$ is undefined. Figure 2 shows an example.

We first consider certain graph problems in $G_{X,Y,\gamma}$ and give the definitions of ordinary and normalized edit distances in terms of these. Let $\mathcal{P}_{X,Y}$ denote the set of all edit paths between X and Y in $G_{X,Y,\gamma}$. $W_\gamma(p)$ denotes the sum of the weights of the arcs in $p \in \mathcal{P}_{X,Y}$ (W_γ is called the *path-weight function* with respect to γ), and $L(p)$ denotes the *path-length function* which gives the number of arcs in $p \in \mathcal{P}_{X,Y}$.

The *shortest path problem* $S_{X,Y,\gamma}$ is the weight-minimization problem such that

$$S^*_{X,Y,\gamma} = \min_{p \in \mathcal{P}_{X,Y}} W_\gamma(p) \tag{2.1}$$

γ	ε	a	b
ε	-	9	9
a	7	0	5
b	7	5	0

FIG. 2. General representation of γ .

where for any optimization problem M, we use M^* to denote its optimum value.

The *ordinary edit distance* $ED^*_{X,Y,\gamma}$ is the weight of an edit sequence with minimum weight. $ED^*_{X,Y,\gamma} = S^*_{X,Y,\gamma}$ when the cost function γ satisfies a pair of conditions: First, γ should not include negative weights, for otherwise $ED^*_{X,Y,\gamma}$ is undefined even though the definition (2.1) yields some value for $S^*_{X,Y,\gamma}$. In the presence of negative weights, some sequence of edit operations may be repeated, undoing the effect of earlier operations, and decreasing the total weight. This situation is similar to the shortest path problem in arbitrary weighted graphs with negative cycles. Second, the triangle inequality must hold for γ: i.e. for all triples of edit operations $s_1 \to s_3$, $s_1 \to s_2$, $s_2 \to s_3$ where $s_1, s_2, s_3 \in \{\epsilon\} \cup \Sigma$,

$$\gamma_{s_1 \to s_3} \leq \gamma_{s_1 \to s_2} + \gamma_{s_2 \to s_3} \ .$$

To see the relevance of this, assume all edit operations have some large weight except for $\gamma_{c \to f} = \gamma_{f \to h} = 0$, and consider the strings “cat” and “hat”. The optimum ordinary edit distance can be achieved by performing two operations on the first index position of “cat” by “cat” $\to$ “fat” $\to$ “hat”. But then the resulting ordinary edit distance is smaller than $S^*_{X,Y,\gamma}$. The problem is that an optimal edit sequence does not correspond to any edit path in $\mathcal{P}_{X,Y}$. An optimal edit sequence can be captured in the set of edit paths only if no multiple editing is allowed on any index position.

Consider the problem of finding the minimum amortized weight of the paths in $\mathcal{P}_{X,Y}$. We denote this problem by $R_{X,Y,\gamma}$. Thus

$$R^*_{X,Y,\gamma} = \min_{p \in \mathcal{P}_{X,Y}} \frac{W_\gamma(p)}{L(p)} \ . \tag{2.2}$$

The *normalized edit distance* problem $NED_{X,Y,\gamma}$, seeks for the minimum amortized weight over all possible edit sequences. $NED^*_{X,Y,\gamma} = R^*_{X,Y,\gamma}$ when the two conditions discussed above for the cost function hold. We assume that the cost function γ is non-negative and it satisfies the triangle inequality, which has been the common practice throughout the literature. Note that if $m = n = 0$ then both $NED^*_{X,Y,\gamma}$ and $R^*_{X,Y,\gamma}$ are undefined.

Figure 3 shows the resulting edit graph $G_{X,Y,\gamma}$ for the strings of figure 1, and the cost function in figure 2. The paths p_1 and p_2 in figure 3 are optimal edit paths for ordinary and normalized edit distances, respectively. Note that the post-normalized value for p_1 is $W_\gamma(p_1)/L(p_1) = 5$, whereas the optimum value of NED as defined in (2.2) is achieved by p_2 with $W_\gamma(p_2)/L(p_2) = 4$.

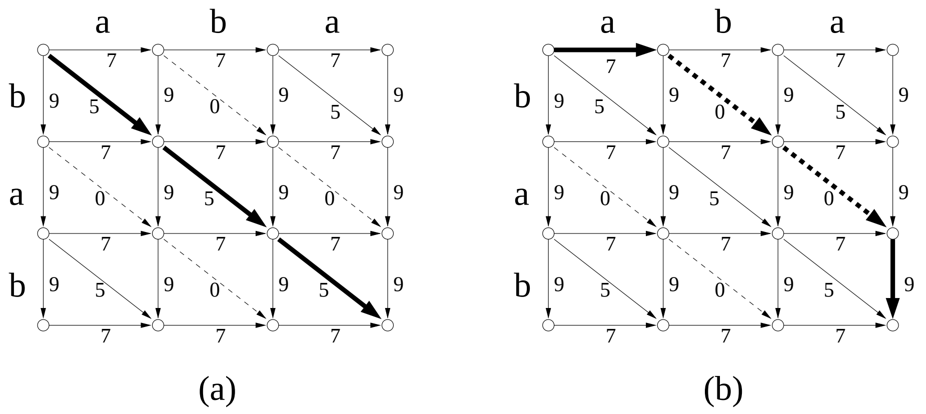

FIG. 3: (a) An optimal edit path p_1 (for ED) with weight 15, $ED^*_{X,Y,\gamma} = 15$. (b) An optimal edit path p_2 (for NED) with amortized weight $16/4 = 4$, $NED^*_{X,Y,\gamma} = 4$.

Given an alphabet Σ, let $< e_1, e_2, \ldots, e_d >$ be an ordering of edit operations where $d = (|\Sigma| + 1)^2 - 1$, the size of γ . For strings X and Y, define $\mathcal{F} = \{(f_{e_1}(p), f_{e_2}(p), \ldots, f_{e_d}(p) \mid p \in \mathcal{P}_{X,Y}\}$ where $f_e(p)$ denotes the number of arcs in p corresponding to the edit operation e . W_γ and L can be thought of as linear functions from $\mathcal{F}$ to the real numbers. For a given $p \in \mathcal{P}_{X,Y}$

$$\begin{aligned} W_\gamma(p) &= \sum_{1\le i\le d} \gamma_{e_i} f_{e_i}(p) \qquad (2.3)\\ L(p) &= \sum_{1\le i\le d} f_{e_i}(p). \end{aligned}$$

Consequently $NED_{X,Y,\gamma}$ can be viewed as the minimization problem whose objective function in (2.2) is the ratio of the two linear functions in (2.3).

3 Optimizing the Ratio of Two Linear Functions

For $z = (z_1, z_2, \ldots, z_n) \in D$ for some domain $D \subseteq \mathbb{R}^n$, let A the problem of minimizing $c_0 + c_1 z_1 + \cdots + c_n z_n$ and let B be the problem of minimizing $(a_0 + a_1 z_1 + \cdots + a_n z_n)/(b_0 + b_1 z_1 + \cdots + b_n z_n)$ where the denominator is assumed to be always positive in D. For λ real, let $A(\lambda)$ denote the *parametric problem* of minimizing $(a_0 + a_1 z_1 + \cdots + a_n z_n) - \lambda(b_0 + b_1 z_1 + \cdots + b_n z_n)$ in D. This is problem A with $c_i = a_i - \lambda b_i$, $i = 0, 1, \ldots, n$.

Problem A: $minimize\ c_0 + c_1 z_1 + \cdots + c_n z_n$
s.t. $z \in D$

Problem B: $minimize\ \frac{a_0+a_1z_1+\cdots+a_nz_n}{b_0+b_1z_1+\cdots+b_nz_n}$
s.t. $z \in D$

Problem $A(\lambda)$: $minimize\ [(a_0 + a_1z_1 + \cdots + a_nz_n) - \lambda(b_0 + b_1z_1 + \cdots + b_nz_n)]$
s.t. $z \in D$

Dinkelbach's algorithm [3] can be used to solve B when a solution for A is available. The algorithm uses the parametric method of an optimization technique known as *fractional programming*. In fact, this method is also applicable to optimization problems involving ratios of functions which are more general than linear. The thesis of the parametric method is that an optimal solution to B can be achieved via the solution of $A(\lambda)$. In fact

$$\lambda = B^* \text{ iff } A^*(\lambda) = 0 \; .$$

That is, λ is the optimum value of problem B iff the optimum value of the parametric problem $A(\lambda)$ is zero.

Dinkelbach's algorithm starts with an initial value for λ and repeatedly solves $A(\lambda)$. At each instance of the parametric problem, an optimal solution z of $A(\lambda)$ yields a ratio for B. This new ratio is either equal to λ, in which case it is optimum, or better (smaller) than λ . If it is equal to λ then the algorithm terminates. Otherwise, the ratio is taken to be the new value of λ and $A(\lambda)$ is solved again. It can be shown that when continued in this fashion, this algorithm takes finitely many steps to find an optimal solution to B if D is a finite set. Furthermore even if D is not finite, convergence to B^* is guaranteed to be superlinear. Various properties of Dinkelbach's algorithm and fractional programming can be found in [2, 3, 6, 14].

Megiddo [9] introduced a general technique to develop an algorithm for B given an algorithm for A. The resulting algorithm for B is the algorithm for A with $c_i = a_i - \lambda b_i$, for $i = 0, 1, \ldots, n$, where λ is treated as a variable, not a constant. That is, the algorithm is the same algorithm as that for A except that the coefficients are not simple constants but linear functions of the parameter λ. Instead of repeatedly solving $A(\lambda)$ with improved values of λ, this alternative solution simulates the algorithm for A over these coefficients. The assumption is that the operations among coefficients in the algorithm for A are limited to comparisons and additions. Additions of linear functions are linear and can be computed immediately, but comparisons among linear functions need to be done with some care. The algorithm needs to keep track of the interval in which the optimum value B^* lies. This is essential because comparisons in the algorithm for A now correspond to those among linear functions, and outcomes may vary depending on interval under consideration for λ.

The algorithm starts with the initial interval $[-\infty, +\infty]$ for B^*. If the functions to be compared intersect, then their intersection point λ' determines two subintervals of the initial interval. In calculating which of the two subintervals contains B^*, algorithm for A is called for help, and problem $A(\lambda')$ is solved. The new interval and the result of the comparison are determined from the sign of the optimum value $A^*(\lambda')$ as will be explained later. With Megiddo's technique, if A is solvable using $O(p(n))$ comparisons and $O(q(n))$ additions then B can be solved in time $O(p(n)(p(n)+q(n)))$. We refer the reader to Megiddo's paper [9] for the details of this approach.

Megiddo also showed that for some problems the “critical values” of λ , values of λ which affect the outcome of comparisons, can be precomputed. In such cases these values give us the possible candidates for the endpoints of the smallest interval which eventually contains the optimum value B^*. Whenever this can be done, *binary search* can be used to find B^* as follows: If $A^*(\lambda) = 0$, then $\lambda = B^*$, and an optimal solution z of $A(\lambda)$ is also an optimal solution of B . On the other hand, if $A^*(\lambda) > 0$, then a larger λ, and if $A^*(\lambda) < 0$, then a smaller λ should be tested (i.e. problem $A(\lambda)$ should be solved with a new value of λ). This procedure continues until the “correct” value B^* is found. Let λ' be the smallest value in the set for which $A^*(\lambda')$ is greater than or equal to zero. Then an optimal solution z of $A(\lambda')$ yields the optimum value B^* of B. Fewer number of invocations of algorithm A may reduce the time complexity of solving B significantly, which is the case in problems such as minimum ratio cycles, and minimum ratio spanning trees [9].

4 Algorithms for NED

We analyze the NED problem assuming two different types of commonly used cost functions: uniform weights and rational weights. In each case, problem A is the shortest path problem $S_{X,Y,\gamma}$ with the objective function in (2.1), and problem B is the normalized edit distance problem $NED_{X,Y,\gamma}$ whose objective function is given in (2.2). We call the parametric problem $A(\lambda)$ the *parametric shortest path* problem whose objective is defined as the following :
For any λ real

$$S^*_{X,Y,\gamma}(\lambda) = \min_{p \in \mathcal{P}_{X,Y}} \left[W_\gamma(p) - \lambda L(p)\right] . \tag{4.1}$$

In the cases of cost functions we analyze, we consider two different formulations for the parametric shortest path problem in terms of the shortest path problem. We apply Megiddo's search technique either on a precomputed set of values, or on a range of values in which the smallest distance between any two distinct values is larger than some threshold δ we precompute.

4.1 Uniform Weights

A *cost function* γ for uniform weights has a simpler representation than the general case. Ordinarily γ specifies the weights of insertion, deletion, and non-matching substitution. Generally the weight of a matching substitution is assumed to be zero. In our case we denote the uniform cost function by γ_u, and represent it as a 4-tuple of real numbers, $\gamma_u = (\gamma_I, \gamma_D, \gamma_M, \gamma_N)$. These specify the weight of an insertion (γ_I), deletion (γ_D), matching substitution (γ_M), and non-matching substitution (γ_N). As an example $\gamma_u = (9, 7, 0, 5)$ is the representation, in this format, of the cost function in figure 2 .

For a given $p \in \mathcal{P}_{X,Y}$, let $h(p)$, $v(p)$, $d_M(p)$, and $d_N(p)$ denote respectively the number of horizontal, vertical, matching diagonal, and non-matching diagonal arcs in

p. W_γ and L can be simplified to the following expressions :

$$\begin{aligned} W_{\gamma_u}(p) &= \gamma_D h(p) + \gamma_I v(p) + \gamma_M d_M(p) + \gamma_N d_N(p), \\ L(p) &= h(p) + v(p) + d_M(p) + d_N(p). \end{aligned} \tag{4.2}$$

Furthermore from the structure of the graph $G_{X,Y,\gamma}$ we have the identities

$$\begin{aligned} m &= h(p) + d_M(p) + d_N(p), \\ n &= v(p) + d_M(p) + d_N(p). \end{aligned}$$

Therefore we can rewrite $W_\gamma(p)$ and $L(p)$ as linear functions of two variables $d_M(p)$ and $d_N(p)$ only, by using the expressions $h(p) = m - d_M(p) - d_N(p)$, and $v(p) = n - d_M(p) - d_N(p)$ in (4.2). In this case, W_{γ_u} and L are linear functions from the set $\{(d_M(p), d_N(p)) \mid p \in \mathcal{P}_{X,Y}\}$ to the real numbers, and the numerator and denominator in (2.2) are given by the linear functions

$$\begin{aligned} W_{\gamma_u}(p) &= m\gamma_D + n\gamma_I \\ &\quad + (\gamma_M - \gamma_I - \gamma_D) d_M(p) + (\gamma_N - \gamma_I - \gamma_D) d_N(p), \\ L(p) &= m + n - d_M(p) - d_N(p). \end{aligned} \tag{4.3}$$

As we show next the set of possible values of λ required for finding NED^*_{X,Y,γ_u} can be precomputed efficiently.

PROPOSITION 4.1
For not both m and n equal to zero, let

$$Q = \{q(r,s) \mid r, s \text{ are non-negative integers, and } r + s \leq \min\{m,n\}\ \}$$

where

$$q(r,s) = \frac{m\gamma_D + n\gamma_I + (\gamma_M - \gamma_I - \gamma_D)r + (\gamma_N - \gamma_I - \gamma_D)s}{m + n - r - s}\ . \tag{4.4}$$

Then

1. $|Q| = O(n^2)$,
2. For any two strings X and Y over Σ of lengths m and n, $\{W_{\gamma_u}(p)/L(p) \mid p \in \mathcal{P}_{X,Y}\} \subseteq Q$,
3. For all $\lambda \in Q$, $\lambda \geq 0$.

That is, the possible amortized weights for the paths in $\mathcal{P}_{X,Y}$ are all included in the set Q whose cardinality is $O(n^2)$ (assuming $m \geq n$), and whose elements are non-negative.

PROOF. Since $r+s \leq \min\{m,n\} = n$, by definition $q(r,s)$ takes on $O(n^2)$ distinct values, proving the result about the size of Q. To see that Q includes all amortized weights note that the expression used to generate the elements of Q in (4.4) uses the same expressions for W_{γ_u} and L in (4.3) except that the variables r, and s replace $d_M(p)$ and $d_N(p)$ respectively, and for any $p \in \mathcal{P}_{X,Y}$, it is necessary that $d_M(p) + d_N(p) \leq \min\{m,n\}$.

Conversely, for non-negative integers r, s with $r+s \leq \min\{m,n\}$, it is easy to see that there are strings $X = x_1 x_2 \cdots x_m$, $Y = y_1 y_2 \cdots y_n$ and an edit path p in $\mathcal{P}_{X,Y}$ such that $d_M(p) = r$, $d_N(p) = s$, and $q(r,s)$ is the amortized weight of the path p provided that not both m and n are zero. Therefore the set Q in the proposition is actually the union of the ratios $\{W_{\gamma_u}(p)/L(p) \mid p \in \mathcal{P}_{X,Y}\}$ where X and Y vary over all strings of length m and n over Σ. Since amortized weights are non-negative for the paths in $P_{X,Y}$, Q does not include a negative number. ∎

PROPOSITION 4.2
For $\lambda \in Q$, the optimum value $S^*_{X,Y,\gamma_u}(\lambda)$ of the parametric shortest path problem can be formulated in terms of the optimum value $S^*_{X,Y,\gamma_u'}$ of a shortest path problem.

PROOF. For any $\lambda \in Q$, using the definition in (4.1), and expressions in (2.1) and (4.3), we have

$$\begin{aligned}
S^*_{X,Y,\gamma_u}(\lambda) &= \min_{p \in \mathcal{P}_{X,Y}} \quad [W_{\gamma_u}(p) - \lambda L(p)] \\
&= \min_{p \in \mathcal{P}_{X,Y}} \quad [\, m\gamma_D + n\gamma_I \\
&\qquad\qquad + (\gamma_M - \gamma_I - \gamma_D) d_M(p) \\
&\qquad\qquad + (\gamma_N - \gamma_I - \gamma_D) d_N(p) \\
&\qquad\qquad - \lambda(\, m + n - d_M(p) - d_N(p)\,)\,] \\
&= (\min_{p \in \mathcal{P}_{X,Y}} \quad [\, m\gamma_D + n\gamma_I \\
&\qquad\qquad + (\gamma_M + \lambda - \gamma_I - \gamma_D) d_M(p) \\
&\qquad\qquad + (\gamma_N + \lambda - \gamma_I - \gamma_D) d_N(p)\,]\,) \\
&\quad - \lambda(m+n) \\
&= S^*_{X,Y,\gamma_u'} - \lambda(m+n)
\end{aligned}$$

where

$$\gamma_u' = (\gamma_I, \gamma_D, \gamma_M + \lambda, \gamma_N + \lambda)\ . \tag{4.5}$$

Thus computing $S^*_{X,Y,\gamma_u}(\lambda)$ involves solving the shortest path problem $S_{X,Y,\gamma_u'}$, and performing some simple arithmetic afterwards. Note that, all $\lambda \in Q$ are non-negative. Therefore γ_u' and $G_{X,Y,\gamma_u'}$ have no negative weights. ∎

Based on this result, we propose the following algorithm `UniformNED` to solve the NED problem with uniform weights: The algorithm first generates the set of numbers Q which includes all possible amortized weights, i.e. potential optimal values of NED as described in proposition 4.1. Next, the optimum value NED^*_{X,Y,γ_u} is sought in this set by simulating a binary search. At each iteration, the median of the current set is found, and with this value a parametric shortest path problem instance is created. This parametric problem is solved by using the formulation in proposition 4.2. If the optimum value of the parametric problem is zero then the median is the optimum value of NED and the algorithm terminates with returning the median. If it is negative then the search needs to be directed to smaller values; otherwise if it is positive, the search space is reduced to larger values. In both cases, the non-optimal half is removed from the set. This process may remove all the elements but one from Q . The remaining element is returned as the optimum value of NED .

The main steps of the algorithm are shown in figure 4. The algorithm is clearly

```
ALGORITHM UniformNED
Step 1 :  If $m = n = 0$ then return($-1$) signalling an undefined result.
Step 2 :  Return trivial answers :
             If $m = 0$ then return($\gamma_I$),
             If $n = 0$ then return($\gamma_D$).
Step 3 :  Generate the set $Q$ (of proposition 4.1).
Step 4 :  While($|Q| > 1$) do
Step 5 :      Find the median $\lambda_{med}$ of $Q$ .
Step 6 :      $v \leftarrow S^*_{X,Y,\gamma_u}(\lambda_{med})$ (use proposition 4.2).
Step 7 :      If $v = 0$ then return($\lambda_{med}$)
Step 8 :      else if $v < 0$ then remove from $Q$
                 $\lambda_{med}$ and the elements larger than $\lambda_{med}$
Step 9 :      else (if $v > 0$ then) remove from $Q$
                 $\lambda_{med}$ and the elements smaller than $\lambda_{med}$ .
Step 10:  End (while)
Step 11:  Return the element in $Q$.
Step 12:  End.
```

FIG. 4. NED algorithm `UniformNED` for uniform weights.

correct when m or n is zero. Otherwise $\{W_{\gamma_u}(p)/L(p) \mid p \in \mathcal{P}_{X,Y}\} \neq \phi$, and the correctness in this case follows from the facts that there exists $\lambda \in Q$ such that $S^*_{X,Y,\gamma_u}(\lambda) = 0$ (take $\lambda = NED^*_{X,Y,\gamma_u}$), and if $S^*_{X,Y,\gamma_u}(\lambda) = 0$ then $\lambda \in \{W_{\gamma_u}(p)/L(p) \mid p \in \mathcal{P}_{X,Y}\}$.

Note that any shortest path algorithm A for $G_{X,Y,\gamma_u{}'}$ with non-negative weights can be used to compute $S^*_{X,Y,\gamma_u{}'}$ as part of the computation of $S^*_{X,Y,\gamma_u}(\lambda)$ in step 6 of algorithm `UniformNED`, provided that $\gamma_u{}'$ as defined in (4.5), does not cause a conflict with algorithm A's assumption on the edge weights of $G_{X,Y,\gamma_u{}'}$.

THEOREM 4.3
If algorithm A computes $S^*_{X,Y,\gamma_u{}'}$ (without the assumption of the triangle inequality for $\gamma_u{}'$) with time complexity $C(m,n)$, then NED^*_{X,Y,γ_u} can be computed in time $O(n^2) + C(m,n)O(\log n)$.

PROOF. We show that this complexity result can be achieved by using algorithm `UniformNED`. Step 3 of the algorithm takes $O(n^2)$ time. The while loop iterates $O(\log n)$ times. If linear-time median finding algorithm [1] is used in step 5, then the total time (from start to completion of the loop) spent in steps 5, 8, and 9 to find the medians, and remove the elements from Q is $O(n^2)$ since the size of Q is halved after each iteration. Solving a parametric problem in step 6, which involves a shortest path computation problem and simple arithmetic, by proposition 4.2, takes $C(m,n)$ time using algorithm A . The remaining steps take constant time. Thus the resulting time complexity is as expressed in the theorem. ∎

Wagner and Fisher's ordinary edit distance algorithm [17], whose time complexity is $O(mn)$ and space complexity is $O(n)$, actually computes $S^*_{X,Y,\gamma_u{}'}$. It can be used as algorithm A to solve the parametric shortest path problems in algorithm `UniformNED`. Since $C(m,n) = O(mn)$ in this case, and the set Q requires $O(n^2)$ space, we have

COROLLARY 4.4
Algorithm `UniformNED` computes the normalized edit distance NED^*_{X,Y,γ_u} using $O(mn \log n)$ operations and $O(n^2)$ space.

4.2 Rational Weights

We first analyze the case when the weights are integral. Even though it does not seem feasible to precompute all possible amortized weights in this case, we will show that an efficient search for an optimum value is still possible by using the fact that the distribution of the amortized weights is not arbitrary.

PROPOSITION 4.5
If Q is the non-empty set of possible values for $NED^*_{X,Y,\gamma}$ i.e.,

$$Q = \{W_\gamma(p)/L(p) \mid p \in \mathcal{P}_{X,Y}\},$$

and $\delta = \delta_Q$ denotes the smallest gap in Q defined by

$$\delta = \min \{ \, |a-b| \; \mid a, b \in Q, a \neq b\},$$

then $\delta \geq \frac{1}{(m+n)^2}$.

PROOF. Let p_1, and p_2 be two edit paths with different amortized weights such that $\delta = W_\gamma(p_1)/L(p_1) - W_\gamma(p_2)/L(p_2)$. Then

$$\delta = \left| \frac{W_\gamma(p_1)L(p_2) - W_\gamma(p_2)L(p_1)}{L(p_1)L(p_2)} \right| \geq \frac{1}{(m+n)^2}$$

follows from the facts that $|W_\gamma(p_1)L(p_2) - W_\gamma(p_2)L(p_1)| \geq 1$ (distance between any two distinct integers), and $L(p_1)L(p_2) \leq (m+n)^2$ since $L(p_1), L(p_2) \leq m+n$. ∎

PROPOSITION 4.6
For λ real and a cost function γ with integral weights, the optimum value $S^*_{X,Y,\gamma}(\lambda)$ of the parametric shortest path problem can be formulated in terms of the optimum value $S^*_{X,Y,\gamma'}$ of a shortest path problem.

PROOF. For any $\lambda \in Q$, using the definition in (4.1), and expressions in (2.1) and (2.3) we have

$$\begin{aligned} S^*_{X,Y,\gamma}(\lambda) &= \min_{p \in \mathcal{P}_{X,Y}} \left[\sum_{1 \le i \le d} \gamma_{e_i} f_{e_i}(p) - \lambda \sum_{1 \le i \le d} f_{e_i}(p) \right] \\ &= \min_{p \in \mathcal{P}_{X,Y}} \left[\sum_{1 \le i \le d} (\gamma_{e_i} - \lambda) f_{e_i}(p) \right] \\ &= S_{X,Y,\gamma'} \end{aligned}$$

where $\gamma'_{e_i} = \gamma_{e_i} - \lambda$ for all i, $1 \le i \le d$. ■

From proposition 4.6, $S^*_{X,Y,\gamma}(\lambda)$ is the optimum value of the shortest path problem in $G_{X,Y,\gamma'}$. This definition of the parametric problem was used in fractional programming normalized edit distance algorithm [16]. We want to emphasize that $G_{X,Y,\gamma'}$ may not lead to a valid instance of ordinary edit distance problem because γ' may include a negative weight.

We propose the following algorithm `IntegerNED` for the NED problem with integer weights : The algorithm first computes the smallest possible gap δ between any two distinct values for $NED_{X,Y,\gamma}$ using the expression in proposition 4.5. The algorithm maintains an interval, $[e, f]$, such that the optimum value of $NED_{X,Y,\gamma}$ lies in $[e\delta, f\delta]$ where e, and f are appropriate integral values. Initially e is set to zero, and f is set to γ_{max}/δ where γ_{max} is the maximum weight in γ . The algorithm iteratively solves a parametric shortest path problem with parameter $k\delta$ where k is the median of integers in $[e, f]$. At each iteration the interval is updated according to the sign of the optimum value of the parametric problem as explained in Megiddo's search technique. The effective search space is the integers in $[e, f]$. Each iteration reduces this space by half. The iterations end whenever the optimum value for the parametric shortest path problem is zero upon which the algorithm terminates by returning the parameter $k\delta$ as the optimum value of $NED_{X,Y,\gamma}$, or whenever there remains no integers between e, and f. In the latter case, the algorithm solves a parametric shortest path problem with parameter $f\delta$. An optimal path for this parametric problem yields the optimum amortized weight (the optimum value of $NED_{X,Y,\gamma}$) with which the algorithm terminates.

The main steps of the algorithm are shown in figure 5.

The invariant for the while loop in step 4 is that $e \le f$, and $NED^*_{X,Y,\gamma}$ is in $[e\delta, f\delta]$. We can prove that it holds by induction on the number of iterations. At the beginning (iteration zero) the invariant is true since e and f are initialized to zero and γ_{max}, respectively, and $NED^*_{X,Y,\gamma}$ is in $[0, \gamma_{max}]$. The proof of the inductive step follows from the discussions of Megiddo's search technique. The algorithm returns the parameter value if during one of the iterations the optimum value of the parametric shortest path problem is zero in which case the algorithm is correct. Otherwise, the while-loop terminates with the following conditions being true: $e \le f$ and $e + 1 \ge f$ ($e = f$ or $e + 1 = f$), and $NED^*_{X,Y,\gamma}$ is in $[e\delta, f\delta]$. Since the minimum distance

```
ALGORITHM IntegerNED
Step $1$ :  If $m = n = 0$ then return($-1$) signalling an undefined result.
Step $2$ :  $\delta \leftarrow \frac{1}{(m+n)^2}$
Step $3$ :  $[e, f] \leftarrow [0, \gamma_{max}(m+n)^2]$
Step $4$ :  While ($e + 1 < f$) do
Step $5$ :        $k \leftarrow \lfloor (e+f)/2 \rfloor$
Step $6$ :        $v \leftarrow S^*_{X,Y,\gamma}(k\delta)$ (use proposition 4.6).
Step $7$ :        if $v = 0$ then return($k\delta$)
Step $8$ :        else if $v < 0$ then $f = k$
Step $9$ :        else $e = k$
Step $10$ :  End {while}
Step $11$ :  Solve $S_{X,Y,\gamma}(f\delta)$ to find the optimum value $S^*_{X,Y,\gamma}(f\delta)$ and
             the length $L(p)$ of the corresponding optimal path $p$ .
Step $12$ :  $W_\gamma(p) \leftarrow S^*_{X,Y,\gamma}(f\delta) + f\delta L(p)$
Step $13$ :  Return$\left(\frac{W_\gamma(p)}{L(p)}\right)$
Step $14$ :  End.
```

FIG. 5. NED algorithm `IntegerNED` for integer weights.

between any two possible distinct values for $NED^*_{X,Y,\gamma}$ is at least δ, there are two cases two consider :

(i) $NED^*_{X,Y,\gamma} = e\delta$ or $NED^*_{X,Y,\gamma} = f\delta$,
(ii) $NED^*_{X,Y,\gamma}$ is in $(e\delta, f\delta)$ (in which case there is only one possible value for $NED^*_{X,Y,\gamma}$ in $(e\delta, f\delta)$).

In both cases, an optimal path p for the parametric shortest path problem with parameter $f\delta$ yields the optimum value of $NED_{X,Y,\gamma}$, because of the fact that each new solution to a parametric problem yields a ratio no larger than the parameter value as pointed out in the description of Dinkelbach's algorithm. Note that in step 12 of the algorithm, after determining the optimum value of the parametric shortest path problem and the length of the corresponding optimal path, we have used the expression in definition (4.1) to determine the weight of the optimal path.

THEOREM 4.7
If algorithm A computes $S^*_{X,Y,\gamma'}$ and the length of the corresponding optimal path (without the assumption of non-negativity for γ') with time complexity $C(m,n)$ and space complexity $U(m,n)$ then provided that all the costs in γ are integral, $NED^*_{X,Y,\gamma}$ can be computed in time $O(C(m,n)\log m)$ using $U(m,n)$ space.

PROOF. We show that this complexity result can be achieved by using algorithm `IntegerNED`. The while loop iterates $O(\log(\gamma_{max}(m+n)^2))$ times because the search space on which binary search is performed is included in the set of integers in the range $[0, \gamma_{max}(m+n)^2]$. Solving the parametric problem in step 6 takes $C(m,n)$ time using algorithm A since it involves a shortest path computation and some simple arithmetic by proposition 4.6. Given that algorithm A computes the length of an optimal path along with the optimum value of the parametric problem, step 11 can be performed within time and space complexities of algorithm A . The remaining steps take constant time. Therefore the resulting time complexity is $O(C(m,n)\log(\gamma_{max}(m+n)^2))$. The space complexity of the algorithm is the same as that of algorithm A since the space required by the steps other than those which involve the parametric shortest path computations is only constant. ■

The ordinary distance algorithm of Wagner and Fisher [17] uses a very simple dynamic programming formulation which can easily be modified such that it keeps track of the length of the optimal path, and it also returns this length in addition to the optimum value of the shortest path problem. This modification does not increase the time and space complexities of the algorithm. This modified algorithm is suitable as algorithm A in `IntegerNED`. Since $C(m,n) = O(mn)$ and $U(m,n) = O(n)$ in this case, we have

COROLLARY 4.8
Algorithm `IntegerNED` computes the normalized edit distance $NED^*_{X,Y,\gamma}$ using $O(mn\log m)$ operations and $O(n)$ space provided that all the costs in γ are integral.

If the weights of edit operations in γ are integral multiples of a fixed real number r, we can use algorithm `IntegerNED` to compute $NED^*_{X,Y,\gamma}$ with the same time complexity as follows: Let γ^{int} be an integral cost function such that

$$\gamma_{e_i} = r\gamma^{int}_{e_i} \tag{4.6}$$

for all i , $1 \le i \le d$. Then the following relation holds by the definition in (2.2)

$$NED^*_{X,Y,\gamma} = rNED^*_{X,Y,\gamma^{int}}.$$

Therefore

COROLLARY 4.9
The normalized edit distance $NED^*_{X,Y,\gamma}$ can be computed in $O(mn\log m)$ time and $O(n)$ space provided that all the costs in γ are rational.

5 Remarks

In some cases, an optimal edit path may also be desired besides the normalized edit distance. This can be accomplished by solving an additional parametric shortest path problem. An optimal edit path for the parametric problem $S_{X,Y,\gamma}(NED^*_{X,Y,\gamma})$ has amortized weight $NED^*_{X,Y,\gamma}$ for any cost function γ . Therefore, following the computation of $NED^*_{X,Y,\gamma}$ it suffices to solve $S_{X,Y,\gamma}(NED^*_{X,Y,\gamma})$ for an optimal edit path. This additional computation can be done in $O(mn)$ space.

In theorems 4.3, and 4.7, we have expressed the time complexities of the algorithms `UniformNED` and `IntegerNED` in terms of the time complexity of a shortest path algorithm A which is actually used to solve the parametric shortest path problems. To improve these time complexities one may want to use faster shortest path algorithms.

For edit graphs, ordinary edit distance algorithms perform better than general shortest path algorithms because of the special structure of the graph. But not every fast ordinary edit distance algorithm is a suitable candidate as algorithm A for the computation of shortest paths in our algorithms. In general, ordinary edit distance algorithms assume that the weights are non-negative and/or the triangle inequality holds for the weights.

Proposition 4.6 gives a general formulation of the parametric shortest path problem in terms of the shortest path problem. The formulation preserves the triangle inequality, but it may introduce negative weights into the cost function. To the best of our knowledge the algorithm in [17] is the fastest algorithm A in the presence of negative weights.

In the case of uniform cost function, we gave a different formulation for the parametric problem in proposition 4.2. By this formulation the triangle inequality may be destroyed, but non-negativity of weights is preserved. Even though there are more candidate algorithms that can be used as algorithm A in this case, the algorithm in [17] turns out to be the only feasible one. Next we discuss the reasons for this.

Ordinary edit distance algorithms which assume a fixed cost function (e.g. $\gamma_u = (1,1,0,1)$) and achieve fast running times by using this constant nature of the weights [15, 10] are evidently not suitable as algorithm A in `UniformNED`. This is because each execution of Step 6 of the algorithm uses a different cost function $\gamma_u{}'$.

Masek and Paterson gave a $C(m,n) = O(mn/\log n)$-time algorithm [8] which can be used as algorithm A here, but it is no longer true that the running time of `UniformNED` is as indicated by theorem 4.3. The assumption in the algorithm of Masek and Paterson is that the weights are integral multiples of a positive real constant r. We can easily show that if the assumption is true for γ_u, then it is also true for $\gamma_u{}'$ that arises in `UniformNED`: Suppose that γ_I, γ_D, γ_M, and γ_N are all integral multiples of a real positive number r and let λ be in Q. If $\lambda = 0$, then the assertion is true for the weights in $\gamma_u{}'$, since $\gamma_u{}' = \gamma_u$ in this case. If $\lambda > 0$ then there exist positive integers a, and b such that $\lambda = ar/b$ because of the way we generate the set Q in proposition 4.1. Therefore in this case, γ_I, γ_D, $\gamma_M + \lambda$, and $\gamma_N + \lambda$ are integral multiples of the positive real number r/b. There appear to be no other obvious choice of the new constant such that the assumption holds for $\gamma_u{}'$. But the algorithm of Masek and Paterson takes time proportional to some function of $1/r$ which normally is absorbed in the "big O", since it is independent of m and n. The discrepancy factor among the magnitude of constants for different parametric problems of `UniformNED` can be in order of m since b has to be, in some cases, as large as the maximum path length $m+n$. Therefore even though it may be possible to achieve $O(mn/\log n)$ with γ_u, solving some parametric problems using this A would take significantly (by about a factor of m) more time.

Ukkonen's $O(dn)$-time output-size sensitive algorithm [15], where d is the actual edit distance, does not seem applicable because it assumes that the weights are non-negative and they fulfill the triangle inequality. Even though there are alternate formulations of a parametric shortest path problem $S_{X,Y,\gamma_u}(\lambda)$ in terms of a shortest path problem in $G_{X,Y,\gamma_u{}'}$ with different cost functions $\gamma_u{}'$, the triangle inequality cannot always be met while simultaneously keeping the weights non-negative. For example, setting $\gamma_u{}' = (\gamma_I - \lambda, \gamma_D - \lambda, \gamma_M - \lambda, \gamma_N - \lambda)$ maintains the triangle inequality by allowing negative weights, or setting $\gamma_u{}' = (\gamma_I, \gamma_D, \gamma_M + \lambda, \gamma_N + \lambda)$ guarantees non-negativity of weights but possibly destroys the triangle inequality.

6 Conclusion

In the absence of theoretical time complexity results for the application of fractional programming to normalized edit distance calculations, we have improved the time complexity of normalized edit distance computation from $O(mn^2)$ to $O(mn \log n)$ when the weights of edit operations are uniform, and to $O(mn \log m)$ when the weights are rational.

The worst-case and the expected time complexity of fractional programming formulation of NED need to be investigated for a better assessment of the quality of the algorithms presented in this paper, keeping in mind that our time complexity bounds are in the worst-case. Real applications may be in favor of fractional programming based NED algorithm because of its easy implementation, observed experimental performance, and the generality of the cost function that can be used. It seems that while our algorithms have improved worst-case time complexities, fractional programming normalized edit distance algorithm may actually be provably faster on the average.

References

[1] M. Blum, R. W. Floyd, R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, August 1973.

[2] B. D. Craven. *Fractional Programming.* Helderman Verlag, Berlin, 1988.

[3] W. Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, March 1967.

[4] Ö. Eğecioğlu and M. Ibel. Parallel algorithms for fast computation of normalized edit distances. *Proceedings. Eighth IEEE Symposium on Parallel and Distributed Processing (SPDP'96)*, pages 496–503, October 1996.

[5] Z. Galil and R. Giancarlo. Data structures and algorithms for approximate string matching. *Journal of Complexity*, 4(1):33–72, March 1988.

[6] T. Ibaraki. Parametric approaches to fractional programs. *Mathematical Programming*, 26(3):345–362, August 1983.

[7] A. Marzal and E. Vidal. Computation of normalized edit distances and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):926–932, September 1993.

[8] W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, February 1980.

[9] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, November 1979.

[10] E. W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

[11] B. J. Oommen and K. Zhang. The normalized string editing problem revisited. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6):669–672, June 1996.

[12] S. V. Rice, H. Bunke, and T. A. Nartker. Classes of cost functions for string edit distance. *Algorithmica*, 18(2):271–280, 1997.

[13] D. Sankoff and J. B. Kruskal. *Time Warps, String Edits, and Macromolecules : The Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, MA, 1983.

[14] M. Sniedovich. *Dynamic Programming.* Marcel Dekker, New York, 1992.

[15] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985.

[16] E. Vidal, A. Marzal, and P. Aibar. Fast computation of normalized edit distances. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(9):899–902, September 1995.

[17] R. A. Wagner and M. J. Fisher. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, January 1974.

Received 15 November 1999.