An Efficient Uniform-Cost Normalized Edit Distance Algorithm

Abdullah N. Arslan and Ömer Eğecioğlu
Department of Computer Science
University of California, Santa Barbara
$\{arslan, omer\}@cs.ucsb.edu$

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 which are of three types: insertion, deletion, and substitution of symbols. The model assumes a given weight function which assigns a non-negative real cost to each of these edit operations. The *amortized weight* for a given edit sequence is the ratio of the total weight of the sequence to its length, and the minimum of this ratio over all edit sequences is the *normalized edit distance* between X and Y. Experimental results suggest that for some applications normalized edit distance is better as a similarity measure than ordinary edit distance, which ignores the length of the sequences. Existing algorithms for the computation of normalized edit distance with provable bounds on the complexity require $O(mn^2)$ operations in the worst-case. In this paper we develop an $O(mn \log n)$ worst-case algorithm for the normalized edit distance problem for uniform weights. In this case the weight of each edit operation is constant within the same type, except substitutions can have different costs depending on whether they are matching or non-matching.

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

1 Introduction

Measuring similarity of strings is a well-known problem in computer science which has 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.

Consider 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, or substitutions of characters), by means of which X is transformed into Y. If we call each such sequence as an *edit sequence*, then the ordinary (conventional) *edit distance problem* (ED) seeks for an edit sequence with minimum total weight over all 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 into account. Two binary strings of length 1000 differing in 1 bit 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". It is tempting to normalize the edit distance by an appropriate method: Marzal and Vidal considered a variant called *normalized edit distance* (NED) between X and Y [8]. 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 possibilities. Marzal and Vidal found that NED yields better results in empirical experiments. However, it seems that the computation of NED requires significantly more work than that 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 [8].

Ordinary edit distance can be computed in $O(mn)$ time [14, 18, 6], or $O(mn/\log n)$ time if the weights are rational [9]. 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 [8, 12, 17] and parallel [5]. Observing that an edit sequence length lies in the range m, and $m+n$ inclusive, an $O(mn^2)$ time dynamic programming algorithm can be developed for this problem [8]. Furthermore, it has been noted that NED can be formulated as a special case of *constrained edit distance* (CED) problems [12]. By adapting the techniques used for CED, NED can be computed in $O(smn)$ time where s is the number of substitutions in an optimal edit sequence [12]. 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* [17]; an iterative method in which an ordinary edit distance problem with varying weights is solved at each iteration. Experimental results on both randomly generated synthetic data, and real applications performed by Vidal, Marzal, and Aibar [17] 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 a theoretical bound for this algorithm seems difficult. However under reasonable probabilistic models, the average complexity of this algorithm can be analyzed [1].

In this paper, we direct our attention to NED computations in the case when the weight function is *uniform*; i.e. the weight of an edit operation does not depend on the symbols involved in the operation, but only on its type. We assume four types of operations: insertion, deletion, matching and non-matching substitutions. The weight of each type of an operation is a nonnegative real number. This and even more restricted cases of weight functions have already been studied in the literature in the context of ordinary edit distances [13, 16, 11].

The previously suggested algorithms for NED do not specialize to a faster than $O(mn^2)$ algorithm when the weight function is uniform. We propose an algorithm `UniformNED` which iteratively solves ordinary edit distance problems to compute the NED. We use the standard algorithm of Wagner and Fisher [18] to solve the ordinary edit distance problems, and make use of a technique developed by Megiddo [10] for the optimization of objective functions which are ratios of linear functions. The worst-case resulting time complexity of our algorithm is $O(mn \log n)$, compared to the best provable complexity of $O(mn^2)$ of NED.

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 optimization of the ratio of two linear functions and Megiddo's method. Section 4 describes our algorithm and gives a proof of its

$O(mn \log n)$ worst-case complexity, and section 5 describes its implementation. This is followed by remarks in section 6 and conclusions in section 7.

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, for $m \geq 0$, and $n \geq 0$, 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 character into X, deleting a character from X, or substituting a character from Σ for a character 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 $s \neq x_i$.

The *edit graph* $G_{X,Y}$ of X and Y is a 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 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 $G_{X,Y}$ are divided into three types corresponding to edit operations:

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

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

An *edit path* in $G_{X,Y}$ is a directed path from $(0,0)$ to (m,n). Steps of an edit path correspond to a sequence of edit operations which transforms X into Y as follows: A horizontal arc $((i-1,j),(i,j))$ corresponds to the deletion of x_i, a vertical arc $((i-1,j-1),(i-1,j))$ corresponds to the insertion of y_j immediately before x_i, and a diagonal arc $((i-1,j-1),(i,j))$ corresponds to substitution of symbol y_j for x_i. In Figure 1, horizontal, vertical, and diagonal arcs are labeled by the initial letters of the corresponding edit operations D, I, and S respectively. Matching diagonal arcs are indicated by dashed lines, whereas non-matching diagonal arcs are straight lines.

In case of *uniform costs*, ordinarily a *weight function* $\gamma = (\gamma_I, \gamma_D, \gamma_S)$ is used to give weights to each of the three edit operations allowed. In our case we use a refinement which is defined by a 4-tuple of non-negative real numbers $\gamma = (\gamma_I, \gamma_D, \gamma_M, \gamma_N)$. These specify the cost of an insertion (γ_I), deletion (γ_D), matching substitution (γ_M), and non-matching substitution (γ_N). This turns $G_{X,Y}$ into a weighted graph in which the weights of vertical, horizontal, matching diagonal, and non-matching diagonal arcs are given by $\gamma_I, \gamma_D, \gamma_M, \gamma_N$, respectively.

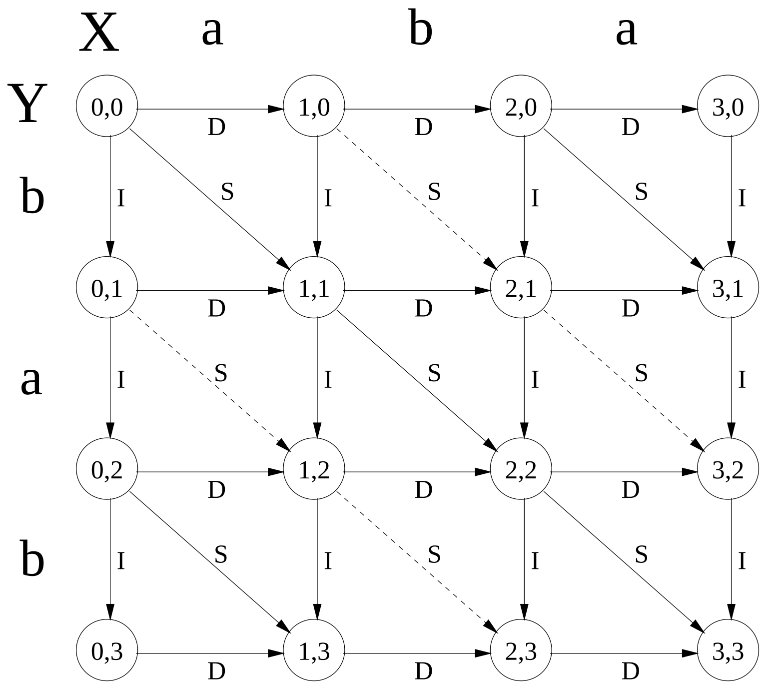

Figure 1: The edit graph $G_{X,Y}$ for the strings $X = aba$ and $Y = bab$. Arcs corresponding to insertion, deletion, and substitution operations are labeled by the letters, I, D, and S, respectively. Non-matching substitutions are dashed diagonal arcs.

The *edit distance* $ED_{X,Y,\gamma}$ is the weight of a least-cost edit path in $G_{X,Y}$. In other words, suppose $\mathcal{P} = \mathcal{P}_{X,Y}$ is the set of all edit paths between X and Y in $G_{X,Y}$. If $W_\gamma(p)$ denotes the sum of the weights of the arcs in $p \in \mathcal{P}$ (W_γ is called the *path-weight function* corresponding to γ), then

$$ED_{X,Y,\gamma} = \min_{p \in \mathcal{P}} W_\gamma(p) \ . \tag{1}$$

The *normalized edit distance* $NED_{X,Y,\gamma}$ is the minimum amortized weight of paths in $\mathcal{P}$. That is, if L denotes the *path-length function* which gives the number of arcs in $p \in \mathcal{P}$, then

$$NED_{X,Y,\gamma} = \min_{p \in \mathcal{P}} \frac{W_\gamma(p)}{L(p)} \ . \tag{2}$$

$NED_{X,Y,\gamma}$ is undefined when $L(p) = 0$ which happens only when both X and Y are null. Figure 2 shows optimal ordinary and normalized edit paths p_1 and p_2 for the graph $G_{X,Y}$ of Figure 1, for $\gamma = (9, 7, 0, 5)$. Note that $W_\gamma(p_1)/L(p_1) = 5$ whereas the optimum value of (2) is $W_\gamma(p_2)/L(p_2) = 4$.

Given strings X and Y, let $\mathcal{F} = \{(h(p), v(p), d_M(p), d_N(p)) \mid p \in \mathcal{P}\}$ where $h(p), v(p), d_M(p)$, 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 defined as linear functions from $\mathcal{F}$ to real numbers. For a given $p \in \mathcal{P}$

$$\begin{aligned} W_\gamma(p) &= \gamma_D h(p) + \gamma_I v(p) + \gamma_M d_M(p) + \gamma_N d_N(p), \qquad (3)\\ L(p) &= h(p) + v(p) + d_M(p) + d_N(p). \end{aligned}$$

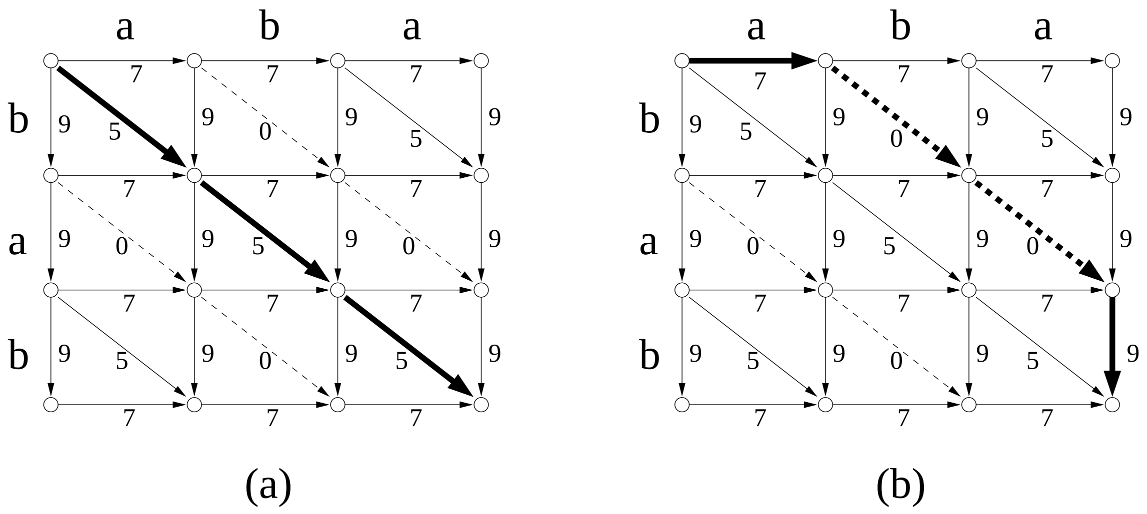

Figure 2: (a) The optimum edit path p_1 with weight 15, $ED_{X,Y,\gamma} = 15$. (b) An optimal edit path p_2 with amortized weight $16/4 = 4$, $NED_{X,Y,\gamma} = 4$.

From the structure of the graph $G_{X,Y}$ we have

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

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 (3). Consequently the NED problem becomes the minimization problem in (2) in which the numerator and the denominator are given by the linear functions

$$\begin{aligned} W_\gamma(p) &= m\gamma_D + n\gamma_I + (\gamma_M - \gamma_I - \gamma_D)d_M(p) + (\gamma_N - \gamma_I - \gamma_D)d_N(p) \\ L(p) &= m + n - d_M(p) - d_N(p) \end{aligned} \tag{4}$$

3 Optimizing the Ratio of Two Linear Functions

For $z = (z_1, z_2, \ldots, z_n) \in D$ for some domain $D \subseteq \mathrm{I\!R}^n$, 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* $\frac{a_0 + a_1 z_1 + \cdots + a_n z_n}{b_0 + b_1 z_1 + \cdots + b_n z_n}$
s.t. $z \in D$

Problem A(λ): *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 [4] can be used to solve B. It uses the parametric method of an optimization technique known as *fractional programming.* This method is applicable to optimization problems involving ratios of functions 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

$$\lambda^* \text{ is the optimum value of } B \text{ iff } \text{ the optimum value of } A(\lambda^*) \text{ is } 0 \ .$$

Dinkelbach's algorithm starts with an initial value for λ and repeatedly solves $A(\lambda)$. At each execution of of the parametric problem, an optimal solution z of $A(\lambda)$ yields a ratio for B which improves the previous value of the objective function. This 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 the optimum value λ^* is guaranteed to be superlinear. Various properties of Dinkelbach's algorithm and fractional programming can be found in [3, 4, 7, 15].

Megiddo [10] 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 $i = 0, 1, \cdots, n$, 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, 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 λ^* of 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 λ^*. If the functions to be compared intersect, then their intersection point λ' determines two subintervals of the given interval. In calculating which of the two subintervals contains λ^*, algorithm for A is called for help, and the problem $A(\lambda')$ is solved. The new interval and the result of the comparison are determined from the sign of the optimum value v of $A(\lambda')$ 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 [10] for the details of this approach.

Megiddo also showed that for some problems the critical values of λ which affect the outcome of comparisons can be precomputed. In such cases the critical values of λ give us the possible candidates for the endpoints of the smallest interval which eventually contains the optimum value λ^*. Whenever this can be done, *binary search* can be used to find λ^* as follows: Suppose that v is the optimum value of $A(\lambda)$. If $v = 0$, then λ is the optimum value of B, and an optimal solution z of $A(\lambda)$ is also an optimal solution of B. On the other hand, if $v > 0$, then a larger λ, and if $v < 0$, then a smaller λ should be tested (problem $A(\lambda)$ should be solved with a different value of λ). This procedure continues until the "correct" value λ^* is found. Let λ' be the smallest value in the set for which the optimum value of $A(\lambda')$ is greater than or equal to 0. Then an optimal solution z of $A(\lambda')$ yields the optimum value λ^* 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.

In the case of edit distances, problem A is the ordinary edit distance problem, and problem B is the problem of NED in view of our formulation (1), (2), and (4). As we state below the set of possible values of λ required for solving $NED_{X,Y,\gamma}$ can be precomputed efficiently.

Proposition 1 *For not both m and n equal to zero, let*

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

where

$$q(r,s) = \frac{m\gamma_D + n\gamma_I + (\gamma_M - \gamma_I - \gamma_D)r + (\gamma_N - \gamma_I - \gamma_D)s}{m+n-r-s}. \tag{5}$$

Then

1. $|Q| = O(n^2)$,
2. *For any two strings X and Y over Σ of lengths m and n,* $\{W_\gamma(p)/L(p) \mid p \in \mathcal{P}_{X,Y}\} \subseteq Q$,
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 $r+s \leq \min\{m,n\} \leq n$, by definition $q(r,s)$ takes on $O(n^2)$ distinct values, proving the result about the size of Q. The proof of inclusion of all amortized weights in Q directly follows from the definitions of W_γ and L. To see that Q does not include a negative number, note that if $0 \leq r+s \leq \min\{m,n\}$ then $m\gamma_D + n\gamma_I + (\gamma_M - \gamma_I - \gamma_D)r + (\gamma_N - \gamma_I - \gamma_D)s = \gamma_D(m-r-s) + \gamma_I(n-r-s) + \gamma_M r + \gamma_N s \geq 0$. Also the denominator is $m+n-r-s > 0$ since not both m and n are zero. □

Suppose we are given two non-negative integers r,s with $r+s \leq \min\{m,n\}$. Then it is easy to see that there are strings $X = x_1x_2\cdots x_m$, $Y = y_1y_2\cdots y_n$ and an edit path p in $\mathcal{P}_{X,Y}$ such that $d_M(p) = r$ and $d_N(p) = s$. Therefore the set Q computed in the proposition is actually the union of the ratios $\{W_\gamma(p)/L(p) \mid p \in \mathcal{P}_{X,Y}\}$ where X and Y vary over all strings of length m and n over Σ.

4 The Algorithm

We propose the following algorithm `UniformNED` to solve the NED problem: The algorithm first generates the set of numbers Q which includes all possible amortized weights, i.e. potential optimal values of $NED_{X,Y,\gamma}$ as described in proposition 1. Next, the optimum value λ^* 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 edit distance problem instance is created. This parametric problem is solved using an ordinary edit distance algorithm. If the optimum value of the parametric problem is 0 then the median is the optimum value of $NED_{X,Y,\gamma}$, 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 either case, the non-optimal half is removed from the set. The search terminates with the smallest value in the set for which the optimum value of the parametric edit distance problem is 0. This final value is returned as the optimum value of $NED_{X,Y,\gamma}$. An essential feature of algorithm `UniformNED` is that the parametric problem created at each iteration is an instance of the ordinary edit distance algorithm for some weight function γ' of non-negative weights.

The main steps of the algorithm are shown in Figure 3. The algorithm is clearly correct when

```
Algorithm UniformNED
Step 1 :  If m = n = 0 then Return(−1) signalling an undefined result.
Step 2 :  Return trivial answers :
            Return(γ_I) if m = 0, Return(γ_D) if n = 0.
Step 3 :  Generate the set Q (of proposition 1).
Step 4 :  While(true) do
Step 5 :       Find the median λ_med of Q.
Step 6 :       Solve ED_{X,Y,γ}(λ_med). Let v be this optimal value.
Step 7 :       If v = 0 then Return(λ_med)
Step 8 :       else if v < 0 then remove from Q the elements larger than λ_med
Step 9 :       else (if v > 0 then) remove from Q the elements smaller than λ_med.
Step 10:  End (While)
Step 11:  End.
```

Figure 3: NED algorithm for uniform weights.

m and/or n is 0; otherwise the correctness follows from the facts that all possible amortized weights are included in Q, and for any $\lambda \in Q$, $ED_{X,Y,\gamma}(\lambda) = 0$ if and only if $\lambda \in \{W_\gamma(p)/L(p) \mid p \in \mathcal{P}_{X,Y}\}$. Note that, in the latter case the termination is guaranteed since $\lambda \in \{W_\gamma(p)/L(p) \mid p \in \mathcal{P}_{X,Y}\} \neq \emptyset$. The algorithm essentially returns the smallest λ such that $ED_{X,Y,\gamma}(\lambda) = 0$, as required by the NED problem.

Proposition 2 *Each parametric edit distance problem $ED_{X,Y,\gamma}(\lambda)$ in algorithm* `UniformNED` *can be formulated in terms of the ordinary edit distance problem.*

Proof For any $\lambda \in Q$, $ED_{X,Y,\gamma}(\lambda)$ is the minimization problem $\min\limits_{p\in\mathcal{P}} [W_\gamma(p) - \lambda L(p)]$. We calculate that

$$\begin{aligned}
& \min_{p\in\mathcal{P}} \; [W_\gamma(p) - \lambda L(p)] \\
= & \min_{p\in\mathcal{P}} \; [m\gamma_D + n\gamma_I + (\gamma_M - \gamma_I - \gamma_D)d_M(p) + (\gamma_N - \gamma_I - \gamma_D)d_N(p) - \lambda(m + n - d_M(p) - d_N(p))] \\
= & \left[\min_{p\in\mathcal{P}} \; (m\gamma_D + n\gamma_I + (\gamma_M + \lambda - \gamma_I - \gamma_D)d_M(p) + (\gamma_N + \lambda - \gamma_I - \gamma_D)d_N(p)\right] - \lambda(m+n) \\
= & \left[\min_{p\in\mathcal{P}} W_{\gamma'}(p)\right] - \lambda(m+n) \\
= & ED_{X,Y,\gamma'} - \lambda(m+n)
\end{aligned}$$

where $\gamma' = (\gamma_I, \gamma_D, \gamma_M + \lambda, \gamma_N + \lambda)$. Since λ is non-negative for all $\lambda \in Q$, γ' includes no negative weights. Hence we have a valid instance γ' of the ordinary edit distance problem. $\square$

Corollary 1 *Any ordinary edit distance algorithm A can be used to compute a parametric problem $ED_{X,Y,\gamma'}$ in algorithm* `UniformNED`, *provided that A can be used to solve both $ED_{X,Y,\gamma}$ and $ED_{X,Y,\gamma'}$.*

Theorem 1 *If parametric edit distance problems $ED_{X,Y,\gamma'}$ are solvable by an algorithm A where $\gamma' = (\gamma_I, \gamma_D, \gamma_M + \lambda, \gamma_N + \lambda)$ for $\lambda \in Q$ and if $C(m,n)$ denotes the time complexity of algorithm A, then $NED_{X,Y,\gamma}$ 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 [2] is used in step 5, then the total time (from start to completion of the loop) spent in steps 8, and 9 is $O(n^2)$. Solving a parametric problem in step 6, which is an ordinary edit distance computation problem by proposition 2, takes $C(m,n)$ time. The remaining steps take constant time. It is easy to see that the resulting time complexity is as expressed in the theorem. □

Wagner and Fisher's $O(mn)$-time edit distance algorithm [18] can be used to compute the least-cost paths in $G_{X,Y}$. This algorithm is suitable as algorithm A in `UniformNED` to solve the parametric ordinary edit distance problems, since the $O(mn)$ time complexity is independent of the weight function. Since $C(m,n) = O(mn)$ in this case, we have

Corollary 2 *Algorithm* `UniformNED` *computes the normalized edit distance $NED_{X,Y,\gamma}$ using $O(mn \log n)$ operations.*

5 Implementation Details

Termination of `UniformNED` requires absolute accuracy in ordinary edit distance computations. Errors associated with floating point operations may be problematic. This can be solved by modifying the algorithm as shown in Figure 4.

```
Algorithm UniformNED2
Step 1 :  If m = n = 0 then Return(−1) signalling an undefined result.
Step 2 :  Return trivial answers :
            Return(γ_I) if m = 0, Return(γ_D) if n = 0.
Step 3 :  Generate the set Q (of proposition 1).
Step 4 :  While(|Q| > 1) do
Step 5 :       Find the median λ_med of Q.
Step 6 :       Solve ED_{X,Y,γ}(λ_med). Let v be this optimal value.
Step 7 :       If v = 0 then Return(λ_med)
Step 8 :       else if v < 0 then remove from Q the elements larger than λ_med,
Step 9 :       else (if v > 0 then) remove from Q the elements smaller than λ_med.
Step 10:  End (While)
Step 11:  Return the element in Q.
Step 12:  End.
```

Figure 4: Modified NED algorithm for uniform weights.

Assuming that there is no sign-error in the result of ordinary edit distance computation, the optimum value will be returned either in step 7 or after the termination of the while loop, since it will not be removed from Q. Therefore termination is guaranteed. The time complexity of `UniformNED2` is the same as that of `UniformNED`.

6 Remarks

There are alternate ways of formulating $NED_{X,Y,\gamma'}$ in terms of least-cost paths if we relax certain conditions. For example, it is possible to write

$$NED_{X,Y,\gamma'} = \min_{p \in \mathcal{P}} W_{\gamma'}(p)$$

where $\gamma' = (\gamma_I - \lambda, \gamma_D - \lambda, \gamma_M - \lambda, \gamma_N - \lambda)$ as used in [17]. However in such a formulation, γ' may include a negative weight, resulting in a non-standard instance of the ordinary edit distance problem that needs to be solved as a part of `UniformNED`.

We have formulated algorithm `UniformNED` in such a way that any ordinary edit distance algorithm A can be used to compute a parametric problem $ED_{X,Y,\gamma'}$ of Step 6, as long as A is capable of solving both $ED_{X,Y,\gamma}$ and $ED_{X,Y,\gamma'}$ as indicated in corollary 1. This property of A needs to be kept in mind in the time complexity $O(n^2) + C(m,n)O(\log n)$ of theorem 1. In other words, not every fast algorithm A for the ordinary edit distance calculation may be a suitable candidate for A, if it is not general enough to meet the conditions in corollary 1. For example edit distance algorithms which assume a fixed weight function (e.g. $\gamma = (1,1,1)$) and achieve fast running times by using this constant nature of the weights [16, 11] are evidently not suitable for `UniformNED`. This is because each execution of Step 6 of the algorithm uses a different weight function γ'.

Masek and Paterson [9] gave a $C(m,n) = O(mn/\log n)$-time algorithm A which can be used here, but it is no longer true that the running time of `UniformNED` is as indicated by theorem 1. 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 λ, then it is also true for λ' that arises in `UniformNED`. Suppose that γ_I, γ_D, γ_M, and γ_N are all integer multiples of a real positive number r and let λ in Q. If $\lambda = 0$, then the assertion is true for the weights in γ', since $\gamma' = \gamma$ in this case. If $\lambda > 0$ the there exist positive integers a, and b such that $\lambda = ar/b$ because of the way we generate the set Q in proposition 1. Therefore in this case, γ_I, γ_D, $\gamma_M + \lambda$, and $\gamma_N + \lambda$ are integer 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 γ'. 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, however. Therefore even though it may be possible to achieve $O(mn/\log n)$ with γ, 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 time algorithm [16], where d is the actual edit distance, does not seem applicable because it assumes the triangle inequality, which basically says that among the paths connecting any two nodes in $G_{X,Y}$, the path with the smallest total weight is also the one with the shortest length. In order for this property to hold for the parametric problems that arise, we need $\gamma' = (\gamma_I + a, \gamma_D + b, \gamma_M + c, \gamma_N + d)$ where the real numbers a, b, c, and d satisfy

$c \leq a + b$, and $d \leq a + b$. These conditions cannot always be met while simultaneously keeping the weights non-negative.

7 Conclusion

In the absence of theoretical time complexity results for the application of fractional programming to normalized edit distance calculations which give good experimental results, 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. Although this result is of theoretical interest, we believe that the real applications will be in favor of fractional programming normalized edit distance algorithm because of its easy implementation and superior experimental performance even without restrictions on the weight function. The worst-case and the expected time complexity of fractional programming formulation of NED need to be investigated for a better assessment of the value of the algorithm presented in this paper, keeping in mind that our $O(mn \log n)$ bound is in the worst-case. It seems that while algorithm `UniformNED` has an improved worst-case time complexity, fractional programming normalized edit distance algorithm may outperform our algorithm on the average. On the other hand the algorithm presented here may benefit from improved algorithms for ordinary edit distance problem, whereas the time complexity of fractional programming based normalized edit distance algorithm can be improved by an improved least-cost path algorithm.

References

[1] A.N. Arslan, Ö. Eğecioğlu. *Average-case Behavior of Fractional Programming for the Normalized Edit Distance*, in preparation.

[2] M. Blum, R.W. Floyd, R. Pratt, R.L. Rivest, R.E. Tarjan. *Time Bounds for Selection.* J. Comput. System Sci., 7:448–461, 1972.

[3] B.D. Craven. "Fractional Programming." Sigma Series in Applied Mathematics 4, Heldermann Verlag, Berlin, 1988.

[4] W. Dinkelbach. *On Nonlinear Fractional Programming.* Management Science, 18:7:492–498, March 1967.

[5] Ö. Eğecioğlu, M. Ibel. *Parallel Algorithms for Fast Computation of Normalized Edit Distances.* Proc. Eighth IEEE Symp. on Parallel and Distributed Processing (SPDP'96), New Orleans, 496–503, October 1996.

[6] Z. Galil, R. Giancarlo. *Data Structures and Algorithms for Approximate String Matching.* Journal of Complexity, 4:33–72, 1988.

[7] T. Ibaraki. *Parametric Approaches to Fractional Programs.* Mathematical Programming, 26:345–362, 1983.

[8] A. Marzal, E. Vidal. *Computation of Normalized Edit Distances and Applications.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 15:9:926–932, September 1993.

[9] W. Masek, M.S. Paterson. *A Faster Algorithm Computing String Edit Distances.* Journal of Computer and System Sciences, 20:18–31, 1980.

[10] N. Megiddo. *Combinatorial Optimization with Rational Objective Functions.* Mathematics of Operations Research, 4:4:414–424, November 1979.

[11] E.W. Myers. *An O(ND) Difference Algorithm and its Variations.* Algorithmica, 1:251–266, 1986.

[12] B.J. Oommen, K. Zhang. *The Normalized String Editing Problem Revisited.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 18:6:669–672, June 1996.

[13] S.V. Rice, H. Bunke, T.A. Nartker. *Classes of Cost Functions for String Edit Distance.* Algorithmica, 18:271–280, 1997.

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

[15] M. Sniedovich. "Dynamic Programming." Pure And Applied Mathematics, A Series of Monographs and Textbooks, Marcel Dekker, New York, 1992.

[16] E. Ukkonen. *Algorithms for Approximate String Matching.* Inform. and Control, 64:100–118, 1985.

[17] E. Vidal, A. Marzal, P. Aibar. *Fast Computation of Normalized Edit Distances.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 17:9:899–902, September 1995.

[18] R.A. Wagner, M.J. Fisher. *The String-to-String Error Correction Problem.* Journal of the Association for Computing Machinery, 21:1,168–173, January 1974.