THE NORMALIZED STRING EDITING PROBLEM REVISITED[+]

B. J. OOMMEN[1] AND K. ZHANG[2]

ABSTRACT

Marzal and Vidal [8] recently considered the problem of computing the *normalized edit distance* between two strings, and reported experimental results which demonstrated the use of the measure to recognize hand-written characters. Their paper formulated the theoretical properties of the measure and developed two algorithms to compute it. In this short communication we shall demonstrate how this measure is related to an auxiliary measure already defined in the literature -- the inter-string *constrained* edit distance [10,11,15]. Since the normalized edit distance can be computed efficiently using the latter, the analytic and experimental results reported in [8] can be obtained just as accurately, but more efficiently, using the strategies presented here.

I. PROBLEM STATEMENT

In the comparison of text patterns, phonemes and biological macromolecules a question that has attracted much interest is that of quantifying the dissimilarity between strings. A review of such distance measures and their applications is in [1,6,12], and in an excellent book edited by Sankoff *et. al.* [13].

The most promising of all measures used to compare strings is the one that relates them using various edit operations [13, pp.37-39]; the edit operations most frequently considered are the deletion, insertion and substitution of individual symbols. This inter-string distance, the Generalized Levenshtein Distance (GLD) is the minimum sum of the edit distances associated with the operations required to edit one string to another. The GLD is closely related to other measures involving the strings, such as their longest common subsequence [2,3]. Algorithms to compute the GLD have been proposed [1,3,4,7,9,13, 14], and its application to the pattern recognition of strings and substrings has been reported [1,5,6,11,12]. Unfortunately, the GLD is inadequate for recognizing noisy subsequences [11] and skeletal images [8].

In [8], Marzal and Vidal showed how an auxiliary related measure could be used in hand-written character recognition. This measure, the *Normalized Edit Distance* (NED) between strings X and Y is the minimum ratio of the sum of the edit distances associated with a sequence of operations editing X to Y to the *number* of operations involved in the particular sequence. Their paper [8] described the properties of the measure (omitted here for brevity) and presented two algorithms to compute it. Apart from deriving the space and time complexities of the algorithms, they also justified the use of the NED over the GLD in pattern recognition applications which processed skeletal silhouetted chain-coded boundaries [8].

In this communication we show how this measure is closely related to an auxiliary measure that has been already defined in the literature - the inter-string *Constrained* Edit Distance (CED). Indeed, the NED can be perceived as a special case of the CED. We also propose algorithms superior to those given in [8].

In terms of notation, $\mathbf{A}$ is the alphabet under consideration and $\mathbf{A}^*$ the set of strings over $\mathbf{A}$. λ $\mathbf{A}$

[+]The work of both the authors was supported by the Natural Sciences and Engineering Council of Canada.

[1]Senior Member, IEEE. Address : School of Computer Science, Carleton University, Ottawa ; CANADA : K1S 5B6.
[2]Member, IEEE. Address : Dept. of Computer Science, University of Western Ontario, London, CANADA : N6A 5B7.

is the null symbol. A string X $\in$ $\mathbf{A}^*$ of the form X = X[1..N] = $x_1 \ldots x_N$, where each $x_i \in \mathbf{A}$, is said to be of length |X| = N. Its prefix of length i will be written as X_i, for $1 \le i \le N$. Uppercase symbols represent strings, and lower case symbols, elements of the alphabet. We define the various inter-string distances in terms of three *inter-symbol* distances which are : (i) $d_s(a,b)$, the distance associated with substituting b for a, a,b $\in$ **A**, (although not mandatory, $d_s(a,a)$ is typically zero), (ii) $d_i(a)$, the distance associated with inserting a $\in$ **A**, and, (iii) $d_e(a)$, the distance associated with deleting (erasing) a $\in$ **A**.

II. CONSTRAINED AND NORMALIZED EDIT DISTANCES

II.1 Insertion-Based Constrained Distances

Let τ be any edit constraint specified in terms of the number and type of operations required to edit a string X to Y. τ can be completely specified as a set of integers which represents the number of any one specific operation (insertions, deletions or substitutions) permitted, as shown in [10,11]. The Constrained Edit Distance between X and Y subject to the constraint τ is written as $CED_\tau(X,Y)$. Note that $CED_\tau(X,Y)$ is infinite if τ is the null set, and $X \ne Y$. To compute $CED_\tau(X,Y)$ a dynamic programming technique involving an auxiliary array has been developed and used in recognizing noisy subsequences [10,11].

Let W(i,e,s) be the CED associated with editing $X_{e+s}=x_1x_2\ldots x_{e+s}$ to $Y_{i+s}=y_1y_2\ldots y_{i+s}$ subject to the constraint that exactly i insertions, e deletions and s substitutions are performed in the process of editing. Then, W(i,e,s) possesses the following properties [10,11].

Result I.

Let W(i,e,s) be defined as above for any two strings X and Y. Then, for all valid indices :

$$W(i,e,s)=\text{Min}\Big[\big\{W(i-1,e,s) + d_i(y_{i+s})\big\},\big\{W(i,e-1,s) + d_e(x_{e+s})\big\},\big\{W(i,e,s-1) + d_s(x_{e+s}, y_{i+s})\big\}\Big]. \quad \rightarrow\rightarrow\rightarrow$$

The computation of $CED_\tau(X,Y)$ from W involves combining the elements of W using τ as follows.

Result II.

If |X|=N and |Y|=M, the quantity $CED_\tau(X,Y)$ is related to the array W(i,e,s) as below :

$$CED_\tau(X,Y) = \underset{i}{\text{Min}}_{\tau} \big[W(i, N-M+i, M-i)\big]. \qquad (1) \quad \rightarrow\rightarrow\rightarrow$$

The quantity we are currently interested in, NED(X,Y) can be computed as a by-product of computing the $CED_\tau(X,Y)$ since :

$$NED(X,Y) = \underset{i}{\text{Min}} \big[W(i, N-M+i, M-i) \,/\, (N+i)\big]. \qquad (2)$$

To compute $CED_\tau(X,Y)$ we compute the array W(.,.,.) in planes parallel to the i-axis. Rather than maintain the entire array, W, we can use four quadratic arrays: (i) Wie: the plane in which s=0, (ii) Wis: the plane in which e=0, (iii) Wes0: the e-s plane for the previous value of i, and, (iv) Wes1: the e-s plane for the current value of i. The algorithm traces the trellis plane by plane retaining only the current one parallel to i=0. Prior to updating Wes0, its component required to yield the NED(X,Y) must be included in an expression of the form of (2). Rather than maintain four arrays, we now show that we can compute NED(X,Y) by maintaining only a **single** array perpendicular to the i-axis.

Instead of storing Wie, Wis, Wes0 and Wes1 we maintain only the values corresponding to the e and s axes and the previous plane perpendicular to the i-axis. From Result I we know that when i, e, and s are positive, the only required values are the previous values for the same point in which i is decremented by unity, and the values at the locations already computed for the same value of i, but for the values of e and s **also** decremented by unity. Thus, in every phase only the rows for the current and previous values of "i" are required; using these, the axes of the current plane and the internal node values can be computed. Consequently, the computation of any element on the Wes1 plane does not require the entire Wes0 array. Since the values of the "i-1" plane persist in the Wes array until they are replaced by values of the "i" plane, the entire computation can be done with a **single** quadratic array, Wes(.,.).

This leads to the following efficient algorithm requiring $O(N^2+M)$ space and $O(N^2M)$ time (if N M) to compute the NED. The space complexity claimed in [8] is $O(N^2)$ but it should be $O(N^2+M)$, because, if $N=\log(M)$, then $O(N^2)=O(\log^2(M))$ is inadequate to store Y. The actual space required by Algorithm A1 is less than what is reported in [8] since we do not store the previous and current arrays.

Algorithm A1 (X, Y)
Input : The strings X and Y. The set of edit distances is assumed global.
Output : The NED(X,Y).
Method :

Wes(0,0) ♦ 0
For s ♦ 1 **to** Min[M, N] **Do**
  Wes(0,s) ♦ Wes (0,s-1) + $d_s(x_s, y_s)$
For e ♦ 1 **to** N **Do**
  Wes(e,0) ♦ Wes(e-1,0) + $d_e(x_e)$
For e ♦ 1 **to** N **Do**
  For s ♦ 1 **to** Min[N-e, M] **Do**
    Wes(e,s) ♦ Min [Wes(e-1,s)+$d_e(x_{e+s})$, Wes(e,s-1)+$d_s(x_{e+s}, y_s)$]
If (N-M) > 0 **Then** NED ♦ Wes(N-M, M)/N **Else** NED ♦ ;
For i ♦ 1 **to** M **Do** **Begin**
  Wes(0,0) ♦ Wes(0,0) + $d_i(y_i)$
  For e ♦ 1 **to** N-M **Do**
    Wes(e,0) ♦ Min [Wes(e,0) + $d_i(y_i)$, Wes(e-1,0) + $d_e(x_e)$]
  For s ♦ 1 **to** Min [N, M-i] **Do**
    Wes(0,s) ♦ Min [Wes(0,s)+$d_i(y_{i+s})$, Wes(0,s-1)+$d_s(x_s,y_{i+s})$]
  For e ♦ 1 **to** N-M **Do**
    For s ♦ 1 **to** Min [N-e, M-i] **Do**
      Wes(e,s) ♦ Min [Wes(e,s-1)+$d_s(x_{e+s},y_{i+s})$, Wes(e-1,s)+$d_e(x_{e+s})$, Wes(e,s)+$d_i(y_{i+s})$]
  If (N-M+i) > 0 **Then** NED ♦ Min [NED, Wes(N-M+i, M-i)/(N+i)]
EndFor
END Algorithm A1

II.2 Substitution-Based Constrained Distances

The CED(X,Y) can also be defined in a completely different way [15]. Let D(s, i, j) be the GLD between X_i and Y_j, where the optimal transformation contains exactly s substitutions, and let τ be the set of substitutions

permitted in the constrained set. Note that 0 s, i N, and 0 j M. Now, the CED $\tau(X,Y)$ is related (in a formulation equivalent to (1)) to the array D(s, i, j) since :

$$CED_{\tau}(X,Y) = \min_{s} \left[D(s, N, M) \right]. \quad (3)$$

Again, the NED can be computed as a by-product of computing D, since the former is merely :

$$NED(X,Y) = \min_{s} \left[D(s, N, M) / (N+M-s) \right].$$

The recursive property of D(.,.,.) is given below.

Result III.

Let D(s, i, j) be defined as above for any two strings X and Y. Then, for all valid indices :

$$D(s, i, j) \leftarrow \min\left[D(s, i-1, j) + d_e(x_i), D(s, i, j-1) + d_i(y_j), D(s-1, i-1, j-1) + d_s(x_i,y_j) \right]. \quad \rightarrow\rightarrow\rightarrow$$

Using the properties of D, we develop a simple algorithm for computing the CED and the NED.

Algorithm B1 (X, Y)
Input : The strings X and Y. The set of edit distances is assumed global.
Output : The CED D(s, N, M) for 0 s N, the NED(X,Y) and the index S which yields the NED.
Method :
Begin
D(0, 0, 0) ← 0
For s ← 1 **to** N **Do**
D(s, 0, 0) ←
For i ← 1 **to** N **Do**
D(0, i, 0)← D(0, i-1, 0) + $d_e(x_i)$
For s ← 1 **to** N **Do**
For i ← 1 **to** N **Do**
D(s, i, 0)←
For j ← 1 **to** M **Do Begin**
D(0, 0, j) ← D(0, 0, j-1) + $d_i(y_j)$
For s ← 1 **to** N **Do**
D(s, 0, j) ←
For i ← 1 **to** N **Do**
D(0, i, j) ← D(0, i-1, j) + $d_e(x_i)$
For s ← 1 **to** N **Do**
For i ← 1 **to** N **Do**
D(s, i, j) ← Min[D(s, i-1, j) + $d_e(x_i)$, D(s, i, j-1) + $d_i(y_j)$, D(s-1, i-1, j-1) + $d_s(x_i,y_j)$]
EndFor
NED(X,Y)← $\min_{s}$ [D(s, N, M) /(N+M-s)]
S ← Index which minimizes [D(s,N,M) /(N+M-s)]
End Algorithm B1

The entire computation can be achieved by successively computing a two-dimensional plane for 0 i,s N for increasing values of j from 0 to M. The time complexity of this algorithm is $O(N^2M)$ and the space complexity is $O(N^2+M)$ if we only want to compute the NED. By a generalization of Hirschberg's technique [2] the actual path can be computed using the same space. To do this we present a space-efficient algorithm for finding D(S,N,M) for a fixed S.

Algorithm B2 (X, Y, S)
Input : The strings X and Y and a fixed integer S. The set of edit distances is assumed global.
Output : The sequence of edit operations done to evaluate D(S,N,M) for a fixed S.
Method :

```
N ♦ |X| ; M ♦ |Y| ;
If (S=0) or (N < 2) or (M < 2) Then
    Output the actual edit sequence
Else
    HalfM ♦ M Div 2                          /* Note that HalfM   M-HalfM   (M+1)/2 */
    Y1[1..HalfM] ♦ Y[1..HalfM]
    Y2[1..M-HalfM] ♦ Y[M..HalfM+1]           /* Y2 is the reverse of Y[HalfM+1..M] */
    XR[1..N] ♦ X[N..1]                       /* XR is the reversed form of X */
    Invoke Algorithm B1 for X and Y1 to determine D(s, i, HalfM) for 0   s   S and 0   i   N.
        Store the result in array D1.
    Invoke Algorithm B1 for XR and Y2 to determine D(s, i, M-HalfM) For 0   s   S and 0   i   N.
        Store the result in array D2.
    /*We now Determine s_o and i_o such that
        D1[s_o, i_o]+D2[S-s_o, N-i_o] is the minimum of D1[s, i]+D2[S-s, N-i] */
    Initialize s_o ♦ 0 and  i_o ♦ 0
    MinVal ♦ D1[0, 0] + D2[S, N]
    For i ♦ 0 to N Do
        For s ♦ 0 to S Do
            m1 ♦ D1[s, i] + D2[S-s, N-i]
            If m1 < MinVal Then
                MinVal ♦ m1
                Update s_o and i_o to s_o♦ s and i_o♦ i
            EndIf
        EndFor
    Recursively Invoke Algorithm B2 (X[1..i_o], Y[1..HalfM], s_o)
    Recursively Invoke Algorithm B2 (X[i_o+1..N], Y[HalfM+1..M], S-s_o)
EndIf
```

End Algorithm B2

Consider the algorithm's space requirement. First of all, we only need O(SN) space for D1(s,i) and D2(s,i) for 0 s S and 0 i N. Furthermore, for the recursive calls, it suffices that we pass the indices of the strings (requiring but constant space) instead of the actual strings themselves. Since the depth of the recursive calls is bounded by O(log(M)) we only need O(log(M) space to maintain the run-time stack. Finally, O(N) and O(M) space is required to globally store X and Y themselves. Therefore the space requirement is O(SN+M). For the time complexity we have T(S, N, M) 4 ·S·N·M since,

$$T(S,N,M) = S\cdot N\cdot M + T(s_o, i_o, HalfM) + T(S-s_o, N-i_o, M-HalfM)$$

$$S\cdot N\cdot M + 4\cdot s_o\cdot i_o\cdot HalfM + 4\cdot(S-s_o)\cdot(N-i_o)\cdot(M-HalfM)$$

$$S\cdot N\cdot M + 3\cdot M\cdot s_o\cdot i_o + 3\cdot M\cdot(S-s_o)\cdot(N-i_o)) \text{ since } M-HalfM \quad 3/4M \text{ when } M \quad 2$$

$$S\cdot N\cdot M + 3\cdot M\cdot S\cdot N = \quad 4\cdot S\cdot N\cdot M.$$

Thus the time[3] complexity is $O(S \cdot N \cdot M)$ $O(N^2 M)$ and the space complexity is $O(S \cdot N + M)$ $O(N^2 + M)$.

The final strategy to determine the NED and the optimal sequence for editing X to Y merely invokes Algorithm B1 to determine S, the index which minimizes $[D(s,N,M) / (N+M-s)]$. Algorithm B2 is then invoked to determine the actual edit operations. The time complexity of our overall strategy is $O(N^2 M)$.

We can get a superior time complexity as follows. Let SS be the number of substitutions in GLD(X,Y). Then for the NED(X,Y), the number of substitutions[4] must be less than or equal to SS. So we first execute the GLD algorithm to determine SS, requiring $O(N \cdot M)$ time. We subsequently invoke B1 up to SS to determine S, and this would require $O(SS \cdot N \cdot M)$ time. Finally, we invoke Algorithm B2 to determine the actual sequence requiring $O(S \cdot N \cdot M)$ time. Note that the overall time is $O(SS \cdot N \cdot M)$which is superior to that reported in [8] and would be more feasible in pattern recognition applications when X and Y are many hundreds of characters long as is typical when processing silhouetted chain-code images.

III. CONCLUSIONS

In [8], the Normalized Edit Distance (NED), a variant of the Generalized Levenshtein Distance, was shown to possess excellent properties in the pattern recognition of hand-written characters. We show that the NED can be computed efficiently using the properties of an auxiliary measure already defined in the literature, namely the *constrained* edit distance [10,11,15]. Consequently, we have here developed superior algorithms to compute the NED. Thus, the analytic results in [8] are special case of our results, and the experimental results reported in [8] can be computed more efficiently using our current strategies. We are currently investigating the use of "functional programming"[5] strategies [16] to compute the more general measure, the constrained edit distance. We also believe that the parametric search paradigm of [17] can be used to solve the NED problem in $O(N \cdot M \cdot (\text{Log Log}(N))^2 \cdot \text{Log}^2(M))$ time and $O(N \cdot M)$ space.

REFERENCES

1. P. A. V. Hall and G.R. Dowling, Approximate string matching, *Comput. Surveys*, 12:381-402 (1980).
2. D. S. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Comm. Assoc. Comput. Mach.*, 18:341-343 (1975).
3. R. L. Kashyap and B. J. Oommen, A common basis for similarity and dissimilarity measures involving two strings, *Internat. J. Comput. Math.*, 13:17-40 (1983).
4. R. L. Kashyap and B. J. Oommen, An effective algorithm for string correction using generalized edit distances -I. Description of the algorithm and its optimality, *Inform. Sci.*, 23(2):123-142 (1981).
5. R. L. Kashyap and B. J. Oommen, The noisy substring matching problem, *IEEE Trans. Software Engg.*, SE-9:365-370 (1983).

[3]The latter argument is only for the sake of the proof. Actually, with some tedious "fine tuning" we can do even better and show that the number of operations required is bounded by $2S \cdot N \cdot M$. Since this would distract from the fundamental contribution of the paper, in the interest of brevity, we have opted not to include it here in this correspondence.

[4] For a sequence of edit operations using more than SS substitutions, the number of operations is less than the number of operations used in the computation of GLD(X,Y). Since the GLD(X,Y) is the minimum cost the latter cost would be greater than or equal to the GLD(X,Y). Therefore the ratio of this sequence to the number of operations is greater than or equal to the corresponding ratio for the GLD(X,Y).

[5]The authors are very grateful to the anonymous referees for their helpful comments and for providing them with the unpublished manuscript on computing the NED using "functional programming".

6. K. Kukich, Techniques for automatically correcting words in text, *ACM Comput. Surveys*, 24:377-439 (1992).
7. A. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, *Soviet Phys. Dokl.*, 10:707-710 (1966).
8. A. Marzal and E. Vidal, Computation of Normalized Edit Distance and Applications, *IEEE Trans. on Pattern Anal. and Mach. Intel.*, PAMI-15:926-932 (1993).
9. W. J. Masek and M. S. Paterson, A faster algorithm computing string edit distances, *J. Comput. System Sci.*, 20:18-31 (1980).
10. B. J. Oommen, Constrained string editing, *Information Sciences*, 40: 267-284 (1986).
11. B. J. Oommen, Recognition of noisy subsequences using constrained edit distances, *IEEE Trans. on Pattern Anal. and Mach. Intel.*, PAMI-9:676-685 (1987). (Corrections in PAMI-10:983-984 (1988)).
12. J. L. Peterson, Computer programs for detecting and correcting spelling errors, *Comm. Assoc. Comput. Mach.*, 23:676-687 (1980).
13. D. Sankoff and J. B. Kruskal, *Time Warps,String Edits and Macromolecules: The Theory and practice of Sequence Comparison*, Addison-Wesley (1983).
14. R. A. Wagner and M. J. Fisher, The string to string correction problem, *J. Assoc. Comput. Mach.*, 21:168-173 (1974).
15. K. Zhang, Constrained string and tree editing distance, *Proc. of the IASTED International Symposium on Machine Learning and Neural Networks*, pp 92-95 (1990).
16. E. Vidal, A. Marzal and P. Aibar, Fast Computation of Normalized Edit Distance. To appear in the *IEEE Trans. on Pattern Anal. and Mach. Intel.*
17. N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms, *J. Assoc. Comput. Mach.*, 30:852-865 (1983).

List of Index Terms :
Sequence Processing, String Editing, Normalized String Distances. Levenshtein Distance.