NORTH-HOLLAND

String Alignment With Substitution, Insertion, Deletion, Squashing, and Expansion Operations*

B. JOHN OOMMEN
School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6

ABSTRACT

Let X and Y be any two strings of finite length. The problem of transforming X to Y using the edit operations of substitution, deletion, and insertion has been extensively studied in the literature. The problem can be solved in quadratic time if the edit operations are extended to include the operation of transposition of *adjacent* characters, and is NP-complete if the characters can be edited repeatedly. In this paper we consider the problem of transforming X to Y when the set of edit operations is extended to include the squashing and expansion operations. Whereas in the squashing operation two (or more) contiguous characters of X can be transformed into a single character of Y, in the expansion operation a single character in X may be expanded into two or more contiguous characters of Y. These operations are typically found in the recognition of cursive script. A quadratic time solution to the problem has been presented. This solution is optimal for the infinite-alphabet case. The strategy to compute the sequence of edit operations is also presented.

1. INTRODUCTION

In the study of the comparison of text patterns, syllables, sound phonemes, and biological macromolecules, a question that has interested researchers is that of quantifying the dissimilarity between two strings. A review of such distance measures and their applications is given by Hall and Dowling [2] and Peterson [16]. We recommend to the reader an excellent book edited by Sankoff and Kruskal [18] which discusses in detail the problem of sequence comparison.

The most promising of all distance measures which compare two strings seems to be the one that relates them using various edit operations [18, pp.

*Partially supported by the Natural Sciences and Engineering Research Council of Canada.

0020-0255/95/$9.50
SSDI 0020-0255(94)00110-W

37–39]. The edit operations most frequently considered are the deletion of a symbol, the insertion of a symbol, and the substitution of one symbol for another [2, 5–11, 13, 15, 16, 18–20]. This distance, referred to as the generalized Levenshtein distance (GLD), between two strings is defined as the minimum sum of the edit costs[1] associated with the edit operations required to transform one string to another. Apart from being a suitable index for comparing two strings, this measure is closely related to other numerical and nonnumerical measures that involve the strings, such as the longest common subsequence (LCS) [3–6, 12, 14] and the shortest common supersequence [12].

Various algorithms to compute this distance have been proposed. The most straightforward algorithm to achieve this has been independently published by many authors (see [18]), but the algorithm is generally associated with Wagner and Fischer [19]. A faster algorithm for the finite-alphabet case (and the unbounded-alphabet case for unit costs) has been invented by Masek and Paterson [13]. For the infinite-alphabet case and arbitrary edit costs, it has been shown that Wagner and Fischer's algorithm is optimal [20]. Related to these algorithms are the ones proposed to compute the LCS of two strings by Hirschberg [3, 4], Hunt and Szymanski [5], and Needleman and Wunsch [14]. Bounds on the complexity of the LCS problem have been given by Aho et al. [1]. In this context, it is noteworthy that techniques similar to those described in [19] have been used in the correction of noisy strings, substrings, and subsequences [2, 8, 10, 11, 16, 21, 22], both when the transmission channel is unrestricted and when the channel is restricted to not making consecutive errors [9]. In this case the dictionary is represented as a trie.

All of the above-mentioned algorithms consider the editing of one string, say X, to transform it to Y, with the edit process being absolutely unconstrained. Sankoff [17] pioneered the study of constrained string editing. His algorithm is an LCS algorithm which involves a specialized constraint that has its application in the comparison of amino acid sequences. Later in [25], Oommen presented the first known solution to the problem of editing X to Y subject to any general edit constraint which could be arbitrarily complex, so long as it is specified in terms of the number and type of edit operations to be included in the optimal edit transformation. Using the fundamental principles of constrained string editing and considering the properties of a noisy channel which can garble transmitted sequences, the first algorithm to correct noisy *subsequences* was presented in [21]. The

*[1]These costs are called distances if they obey metric properties such as the triangular inequality. Note that the GLD obeys the triangular inequality even when the individual operation costs do not.

accuracy of the algorithm to correct long subsequences with low "signal-to-noise" ratios was demonstrated in [21, 22], and a related algorithm has also been applied in encryption [23].

Research in editing typically approaches the problem from two distinct perspectives. In the first, the problem is one of finding a minimum cost *series* of edit operations transforming one string into the other. But if a character can be edited at most once in this series, a restricted form of editing results, usually called an alignment. Indeed, when the costs on the edit operations obey a triangular inequality, finding an optimal alignment is equivalent to finding an optimal series, since transforming a character several times is necessarily more costly than transforming a character once. Generally, except for the very first papers on string editing, the literature has not made an issue of the distinction between computing an optimal alignment and computing an optimal series, or between assuming the triangle inequality and allowing general operation costs. This informality is permissible because, when the edit operations are restricted to insertion, deletion, and substitution of characters, an algorithm that computes a minimum cost alignment can be used to solve the minimum cost edit series problem even when the triangle inequality does not hold—given appropriate preprocessing. The preprocessing constructs a new set of costs that do obey the triangle inequality by determining, for every pair (a, b) of characters, the minimum cost of a series to transform a to b. This can be carried out with an all-pair shortest path computation on a graph whose vertices correspond to characters in the alphabet, and whose edges are weighted by the original edit costs.

In all the above-mentioned results, the types of edit operations (or garbling operations if the transmission channel is modelled as a garbling mechanism) are the well-known substitution, insertion, and deletion operations. To our knowledge, there are only few reported papers which study the case when the set of edit operations is expanded [24, 26–28]. In [24], apart from the latter three operations, the set of edit operations has been expanded to also include the transposition operation. The string editing problem with transposition of adjacent characters is NP-complete. When the problem is restricted to series in which any character is edited at most once (this reduces to finding a minimum cost alignment), the problem can be solved in quadratic time. The complexity of the string editing problem with transposition of *nonadjacent* characters is open.

As opposed to [24], in this paper we consider the problem of editing X to Y when the set of edit operations is extended to include the squashing and expansion operations. Whereas in the squashing operation two (or more) contiguous characters of X can be transformed into a single character of Y, in the expansion operation a single character in X may be expanded into two or more contiguous characters of Y. These extensions are applicable in the recognition of cursive script. This is because, in cursive script processing, various squashing and expansion scenarios are encountered: It is not uncommon for the letter "y" to be mistaken for the combination of the characters "ij" and vice versa, and similarly, it is not uncommon for the letter "w" to be mistaken for the combination of the characters among which are "ui" or "iu" and vice versa. Similar examples of squashing and expansion are encountered in applications when the demarcation between the boundaries of the individual symbols is not apparent, as in the recognition of handwriting and phoneme sequences [18]. Indeed, in that sense, our result is a *generalization* more in the flavor of [26] where the expansion and the squashing do not necessarily have to involve the same character. Our work is similar to the excellent results catalogued in [27], except that we are more interested in the alignment problem as opposed to the problem of processing a *series* of edit operations. Thus, we would require a rather straightforward triangular inequality which ensures that a sequences of edit operations that can be effected by a single operation does not have a lesser cost than the single operation itself. The question of how our algorithms can be optimized in the framework of [27] remains open.

Without a triangle inequality, however, finding a minimum cost series of edit operations, with squashing and expansion, is in fact NP-complete. To see this, note that we can accomplish a transposition of adjacent symbols x, y by a squashing operation $xy \rightarrow a_{xy}$ followed by an expansion operation $a_{xy} \rightarrow yx$, where a_{xy} is a new alphabet symbol. Let us suppose that we now assign these squashing and expansion operations the cost 0.5, all other squashing operations infinite cost, and the insertion, deletion, and substitution operations a cost of unity. Then, an algorithm that finds a minimum cost edit series with squashing and expansion operations will find a minimum cost edit series with transposition of adjacent symbols, which is NP-complete [28].

In this paper, we present a quadratic time solution to the problem of string alignment for the expanded set of edit operations. As a corollary to [29], our solution is optimal for the infinite-alphabet case. The technique to compute the optimal sequence of edit operations is also presented. Also, throughout this paper, we shall consider the squashing and expansion operations to be such that two contiguous symbols of one string can be transformed into a single symbol of the second. The case when multiple contiguous symbols (more than two) of one string can be transformed into a single symbol of the other can be generalized from the principles described here.

1.1. *NOTATION*

Let $\mathbf{A}$ be any finite alphabet, and $\mathbf{A}^*$ be the set of strings over $\mathbf{A}$. θ, the null symbol ($\theta \notin \mathbf{A}$), is distinct from μ, the empty string. Let $\widetilde{\mathbf{A}} = \mathbf{A} \cup \{\theta\}$. $\widetilde{\mathbf{A}}$ is referred to as the *appended alphabet*. A string $X \in \mathbf{A}^*$ of the form $X = x_1 \ldots x_N$, 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 $1 \leq i \leq N$. Uppercase symbols represent strings, and lowercase symbols, elements of the alphabet under consideration.

Let Z' be any element in $\widetilde{\mathbf{A}}^*$, the set of strings over $\widetilde{\mathbf{A}}$. The *compression operator* $\mathfrak{C}$ is a mapping from $\widetilde{\mathbf{A}}^*$ to $\mathbf{A}^*$: $\mathfrak{C}(Z')$ is Z' with all occurrences of the symbol θ removed. Note that $\mathfrak{C}$ preserves the order of the non-θ symbols in Z'. For example, if $Z' = f\theta o\theta r$, $\mathfrak{C}(Z') = for$.

1.2. *THE ELEMENTARY EDIT DISTANCES*

As mentioned earlier, throughout this paper, we shall only consider the case when the squashing and expansion operations involve transforming two contiguous symbols of one string into a single symbol of the other. Bearing this in mind, we now define the costs associated with the individual edit operations. If $\mathbf{R}^+$ is the set of nonnegative real numbers, we define the elementary edit distances using five elementary functions $d_s(\cdot,\cdot)$, $d_i(\cdot)$, $d_e(\cdot,\cdot)$, $d_{sq}(\cdot,\cdot)$, and $d_{ex}(\cdot,\cdot)$ defined as follows:

(i) $d_s(p,q)$ is a map from $\mathbf{A} \times \mathbf{A} \to \mathbf{R}^+$ and is called the substitution map. In particular, $d_s(a,b)$ is the distance associated with substituting b for a, $a,b \in \mathbf{A}$. For all $a \in \mathbf{A}$, $d_s(a,a)$ is generally assigned the value zero, although this is not mandatory.

(ii) $d_i(\cdot)$ is a map from $\mathbf{A} \to \mathbf{R}^+$ and is called the insertion map. The quantity $d_i(a)$ is the distance associated with inserting the symbol $a \in \mathbf{A}$.

(iii) $d_e(\cdot)$ is a map from $\mathbf{A} \to \mathbf{R}^+$ and is called the deletion or erasure map. The quantity $d_e(a)$ is the distance associated with deleting (or erasing) the symbol $a \in \mathbf{A}$.

(iv) $d_{sq}(\cdot,\cdot)$ is a map from $\mathbf{A}^2 \times \mathbf{A} \to \mathbf{R}^+$ called the squashing map. The quantity $d_{sq}(ab,c)$ is the distance associated with squashing the string ab into a single character c, where $a,b,c \in \mathbf{A}$.

(v) $d_{ex}(\cdot,\cdot)$ is a map from $\mathbf{A} \times \mathbf{A}^2 \to \mathbf{R}^+$ called the expansion map. The quantity $d_{ex}(c,ab)$ is the distance associated with expanding the character c into the string ab, where $a,b,c \in \mathbf{A}$.

1.3. THE SET OF EDIT POSSIBILITIES: $\Gamma_{X,Y}$

For every pair (X,Y), $X,Y \in A^*$, the finite set $\Gamma_{X,Y}$ is defined by means of the compression operator $\mathfrak{C}$, as a subset of $\widetilde{\mathbf{A}}^* \times \widetilde{\mathbf{A}}^*$, as

$$\Gamma_{X,Y} = \{(X',Y') \mid (X',Y') \in \widetilde{\mathbf{A}}^* \times \widetilde{\mathbf{A}}^*, \text{ and each } (X',Y') \text{ obeys}$$

(i) $\mathfrak{C}(X') = X, \quad \mathfrak{C}(Y') = Y,$
(ii) $|X'| = |Y'|,$
(iii) For all $1 \leq i \leq |X'|$, it is not the case that $x'_i = y'_i = \theta\}$. (1)

By definition, if $(X',Y') \in \Gamma_{X,Y}$, then $\text{Max}(|X|,|Y|) \leq |X'| = |Y'| \leq |X| + |Y|$.

Viewed from the perspective of the three elementary operations, the meaning of the pair $(X',Y') \in \Gamma_{X,Y}$ is interesting. Indeed, every element in $\Gamma_{X,Y}$ corresponds to one way of transforming X into Y, using the edit operations of substitution, deletion, and insertion. The edit operations themselves are specified for all $1 \leq i \leq |X'|$ by (x'_i, y'_i), which represents the transformation of x'_i to y'_i. The cases below consider the three edit operations individually:

(i) If $x'_i \in \mathbf{A}$ and $y'_i \in \mathbf{A}$, it represents the substitution of y'_i for x'_i.
(ii) If $x'_i \in \mathbf{A}$ and $y'_i = \theta$, it represents the deletion of x'_i.
(iii) If $x'_i \in \theta$ and $y'_i \in \mathbf{A}$, it represents the insertion of y'_i.

$\Gamma_{X,Y}$ is an exhaustive enumeration of the set of all the ways by which X can be transformed to Y using these three elementary edit operations where a symbol which is obtained by an edit operation is not subsequently edited. However, on examining the individual elements of $\Gamma_{X,Y}$, it becomes clear that each pair contains more information than that. Indeed, in each pair, there is also information about the various ways by which X can be edited to Y even if the set of edit operations is grown so as to include squashing and expansion. Thus, when $(X',Y') = (ab\theta, cde)$, apart from the operations described above, the pair also represents the substitution of "a" by "c" and the expansion of "b" by "de." Observe that the transformation of a symbol $a \in \mathbf{A}$ to itself is also considered as an operation in the arbitrary pair $(X',Y') \in \Gamma_{X,Y}$. Finally, note that the same set of edit operations (alignment) can be represented by multiple elements in $\Gamma_{X,Y}$. This duplication serves as a powerful tool in the proofs of various analytic results [6, 7, 9, 10, 21, 25].

EXAMPLE 1. Let $X = f$ and $Y = go$. Then,

$$\Gamma_{X,Y} \, \{(f\theta, go), (\theta f, go), (f\theta\theta, \theta go), (\theta f\theta, g\theta o), (\theta\theta f, go\theta)\}.$$

In particular the pair $(\theta f, go)$ represents the edit operations of inserting the "g" and replacing the "f" by an "o." It also represents the expansion of "f" to "go."

Since the edit distance between X and Y is the minimum of the sum of the edit distances associated with operations required to change X to Y, this distance $D(X, Y)$ has the expression

$$D(X,Y) = \operatorname*{Min}_{((X',Y')\in\Gamma_{X,Y})} \left[\sum_{i=1}^{|J'|} [\text{distances associated with the operations in } (X',Y')] \right], \tag{2}$$

where (X', Y') represents J' possible edit operations.

2. THE RECURSIVE PROPERTIES OF THE EDIT DISTANCE

Let $D(X, Y)$ be the edit distance associated with transforming X to Y with the edit operations of substitution, insertion, deletion, squashing, and expansion. In this section, we shall describe how $D(\cdot,\cdot)$ can be computed. To achieve this, we shall first derive the properties of $D(X, Y)$ which can be derived recursively in terms of the corresponding quantities defined for the prefixes of X and Y (X_i and Y_j, respectively), with the assumption that $D(\mu, \mu)$ is zero. Indeed, in this case, we first claim the following straightforward results. They can be proved in the identical way in which the analogous results are proved for the edit distance which entails only the three elementary edit operations [6, 7, 9, 10, 19, 21, 25]. They can also be proved by straightforward enumeration.[2]

LEMMA 0a. *Let $X = X_i = x_1 \ldots x_i$ be the prefix of X and $Y = \mu$, the null string. Then $D(X_i, \mu)$ obeys*

$$D(X_i, \mu) = D(X_{i-1}, \mu) + d_e(x_i).$$

LEMMA 0b. *Let $X = \mu$, and $Y_j = y_1 \ldots y_j$ be the prefix of Y. Then $D(\mu, Y_j)$ obeys*

$$D(\mu, Y_j) = D(\mu, Y_{j-1}) + d_i(y_j).$$

*[2]All the following lemmas can be combined as special cases of Theorem 1. We have separated them just to distinguish the various cases encountered in implementing the algorithm.

LEMMA 0c. *Let $X = x_1$ and $Y = y_1$. Then $D(X,Y)$ obeys*

$$D(X,Y) = \mathrm{Min}\left[D(\mu,Y) + d_e(x_1),\ D(X,\mu) + d_i(y_1),\ d_s(x_1,y_1)\right].$$

LEMMA 0d. *Let $X_i = x_1 \dots x_i$ with $i \geq 2$ be the prefix of X, and $Y = y_1$, the string consisting of the first character of Y. Then $D(X_i,Y)$ obeys*

$$\begin{aligned} D(X_i,Y) = \mathrm{Min}[&D(X_{i-1},Y) + d_e(x_i), D(X_i,\mu) + d_i(y_1), \\ &D(X_{i-1},\mu) + d_s(x_i,y_1), D(X_{i-2},\mu) + d_{sq}(x_{i-1}x_i,y_1)]. \end{aligned}$$

LEMMA 0e. *Let $X = x_1$ be the string consisting of the first character of X and $Y = y_1 \dots y_j$ be the prefix of Y with $j \geq 2$. Then $D(X,Y_j)$ obeys*

$$\begin{aligned} D(X,Y_j) = \mathrm{Min}[&D(\mu,Y_j) + d_e(x_1), D(X,Y_{j-1}) + d_i(y_j), \\ &D(\mu,Y_{j-1}) + d_s(x_1,y_j), D(\mu,Y_{j-2}) + d_{ex}(x_1,y_{j-1}y_j)]. \end{aligned}$$

We shall now state and prove the main result of our paper.

THEOREM 1. *Let $X_i = x_1 \dots x_i$ and $Y_j = y_1 \dots y_j$ with $i, j \geq 2$. Also, let $D(X_i,Y_j)$ be the edit distance associated with transforming X_i to Y_j with the edit operations of substitution, insertion, deletion, squashing, and expansion. Then the following is true:*

$$\begin{aligned} D(X_i,Y_j) = \mathrm{Min}[&D(X_{i-1},Y_j) + d_e(x_i), D(X_i,Y_{j-1}) + d_i(y_j), \\ &D(X_{i-1},Y_{j-1}) + d_s(x_i,y_j), D(X_{i-2},Y_{j-1}) \\ &+ d_{sq}(x_{i-1}x_i,y_j), D(X_{i-1},Y_{j-2}) + d_{ex}(x_i,y_{j-1}y_j)]. \end{aligned}$$

Sketch of Proof: Let Γ_{X_i,Y_j} be the set of all ways by which X_i can be edited into Y_j defined as in (1) for X_i and Y_j. Consider the distance $D(X_i,Y_j)$, which has the expression

$$D(X_i,Y_j) = \underset{((X_i',Y_j')\in\Gamma_{X,Y})}{\mathrm{Min}} \left[\sum_{i=1}^{|J'|} [\text{distances associated with operations in } (X_i',Y_j')]\right],$$

where $(X'_i, Y'_j) \in \Gamma_{X_i,Y_j}$ represents J' possible edit operations. Throughout this proof,[3] we shall assume that the arbitrary element $(X'_i, Y'_j) \in \Gamma_{X_i,Y_j}$ is of length L and is of the form given as

$$X'_i = x'_{i1}x_{i2}\dots x'_{iL}, \quad \text{and} \quad Y'_j = y'_{j1}y'_{j2}\dots y'_{jL}.$$

The proof itself is now tedious and involves partitioning the set Γ_{X_i,Y_j} into nine mutually exclusive and exhaustive subsets as follows:

$$\begin{aligned}
\Gamma^1_{X_i,Y_j} &= \{(X'_i, Y'_j) \mid (X'_i, Y'_j) \in \Gamma_{X_i,Y_j}, \text{ with } x'_{iL-1} \\
&= \theta, x'_{iL} = \theta, y'_{jL-1} = y_{j-1}, y'_{jL} = y_j\}, \\
\Gamma^2_{X_i,Y_j} &= \{(X'_i, Y'_j) \mid (X'_i, Y'_j) \in \Gamma_{X_i,Y_j}, \text{ with } x'_{iL-1} \\
&= \theta, x'_{iL} = x_i, y'_{jL-1} = y_j, y'_{jL} = \theta\}, \\
\Gamma^3_{X_i,Y_j} &= \{(X'_i, Y'_j) \mid (X'_i, Y'_j) \in \Gamma_{X_i,Y_j}, \text{ with } x'_{iL-1} \\
&= \theta, x'_{iL} = x_i, y'_{jL-1} = y_{j-1}, y'_{jL} = y_j\}, \\
\Gamma^4_{X_i,Y_j} &= \{(X'_i, Y'_j) \mid (X'_i, Y'_j) \in \Gamma_{X_i,Y_j}, \text{ with } x'_{iL-1} \\
&= x_i, x'_{iL} = \theta, y'_{jL-1} = \theta, y'_{jL} = y_j\}, \\
\Gamma^5_{X_i,Y_j} &= \{(X'_i, Y'_j) \mid (X'_i, Y'_j) \in \Gamma_{X_i,Y_j}, \text{ with } x'_{iL-1} \\
&= x_i, x'_{iL} = \theta, y'_{jL-1} = y_{j-1}, y'_{jL} = y_j\}, \\
\Gamma^6_{X_i,Y_j} &= \{(X'_i, Y'_j) \mid (X'_i, Y'_j) \in \Gamma_{X_i,Y_j}, \text{ with } x'_{iL-1} \\
&= x_{i-1}, x'_{iL} = x_i, y'_{jL-1} = \theta, y'_{jL} = \theta\}, \\
\Gamma^7_{X_i,Y_j} &= \{(X'_i, Y'_j) \mid (X'_i, Y'_j) \in \Gamma_{X_i,Y_j}, \text{ with } x'_{iL-1} \\
&= x_{i-1}, x'_{iL} = x_i, y'_{jL-1} = \theta, y'_{jL} = y_j\}, \\
\Gamma^8_{X_i,Y_j} &= \{(X'_i, Y'_j) \mid (X'_i, Y'_j) \in \Gamma_{X_i,Y_j}, \text{ with } x'_{iL-1} \\
&= x_{i-1}, x'_{iL} = x_i, y'_{jL-1} = y_j, y'_{jL} = \theta\}, \\
\Gamma^9_{X_i,Y_j} &= \{(X'_i, Y'_j) \mid (X'_i, Y'_j) \in \Gamma_{X_i,Y_j}, \text{ with } x'_{iL-1} \\
&= x_{i-1}, x'_{iL} = x_i, y'_{jL-1} = y_{j-1}, y'_{jL} = y_j\},
\end{aligned}$$

The proof now involves minimizing the terms over each of these sets.

*[3]This notation is not religiously correct. Indeed, the length of the arbitrary element in Γ_{X_i,Y_j} should be $L(X'_i, Y'_j)$. But this will make an already tedious notation even more cumbersome. We request the reader to permit us this breach in notation with the understanding that he remembers that L is dependent on the element itself.

We shall go through the mechanics of minimizing over $\Gamma^1_{X_i,Y_j}$. In every pair in $\Gamma^1_{X_i,Y_j}$, we know that the last two elements of each string in the pair are

$$x'_{iL-1} = \theta, \quad x'_{iL} = \theta, \quad y'_{jL-1} = y_{j-1}, \quad y'_{jL} = y_j.$$

Hence,

$$\min_{\left((X'_i,Y'_j)\in\Gamma^1_{X_i,Y_j}\right)} \sum_{i=1}^{|J'|} [\text{distances associated with operations in } (X'_i, Y'_j)] \tag{3}$$

$$= \min_{\left((X'_i,Y'_j)\in\Gamma^1_{X_i,Y_j}\right)} \sum_{i=1}^{|J'|} [\text{distances associated with operations in } (X'_{iL-1}, Y'_{jL-1})] + d_i(y'_{jL}). \tag{4}$$

For every element in $\Gamma^1_{X_i,Y_j}$, there is a unique element in $\Gamma_{X_i,Y_{j-1}}$ and vice versa, where $\Gamma_{X_i,Y_{j-1}}$ is the set of all ways by which X_i can be transformed into Y_{j-1} defined as in (1) for X_i and Y_{j-1}. This unique element is obtained by merely reducing the length of the strings X'_i and Y'_j by unity. By the inductive hypothesis, the first term in (4) is exactly $D(X_i, Y_{j-1})$. Since $y'_{jL} = y_j$, this tells that the above expression simplifies to

$$D(X_i, Y_{j-1}) + d_i(y_j).$$

In an analogous way, the following result for the other eight minimizations:

minimizing over	$\Gamma^2_{X_i,Y_j}$	leads to	$D(X_{i-1}, Y_j) + d_e(x_i)$,
minimizing over	$\Gamma^3_{X_i,Y_j}$	leads to	$D(X_{i-1}, Y_{j-1}) + d_s(x_i, y_j)$ and
			$D(X_{i-2}, Y_{j-1}) + d_{ex}(x_i, y_{j-1}y_j)$,
minimizing over	$\Gamma^4_{X_i,Y_j}$	leads to	$D(X_i, Y_{j-1}) + d_i(y_j)$,
minimizing over	$\Gamma^5_{X_i,Y_j}$	leads to	$D(X_i, Y_{j-1}) + d_i(y_j)$ and
			$D(X_{i-1}, Y_{j-2}) + d_{ex}(x_i, y_{j-1}y_j)$,
minimizing over	$\Gamma^6_{X_i,Y_j}$	leads to	$D(X_{i-1}, Y_j) + d_e(x_i)$,
minimizing over	$\Gamma^7_{X_i,Y_j}$	leads to	$D(X_{i-1}, Y_{j-1}) + d_s(x_i, y_j)$ and
			$D(X_{i-2}, Y_{j-1}) + d_{sq}(x_{i-1}x_i, y_j)$,
minimizing over	$\Gamma^8_{X_i,Y_j}$	leads to	$D(X_{i-1}, Y_j) + d_e(x_i)$ and
			$D(X_{i-2}, Y_{j-1}) + d_{sq}(x_{i-1}x_i, y_j)$,
minimizing over	$\Gamma^9_{X_i,Y_j}$	leads to	$D(X_{i-1}, Y_{j-1}) + d_s(x_i, y_j)$.

Combining these minimizations proves the theorem.

A note about the *modus operandus* of the proof of Theorem 1 is not out of place. Our result is not merely a direct application of dynamic programming to the current problem, for there is a very fine point in which our proof differs from the proofs currently described in the literature. Indeed, the fundamental difference is that in the current proof, whenever the set over which the minimization is achieved is grown, it is not merely a single optimization scenario which is encountered. Thus in Case 3 of the proof, there are two possible scenarios by which the minimization can be achieved. The first of the scenarios appears again in the processing of Case 7 and in the processing of Case 9. The second appears again in the processing of Case 5. Thus the *same five terms* appear in their different combinations in various cases encountered in the minimization process. This makes our proof more interesting and a trifle more "intriguing" and different from the proof of [19, 27]. Rather, the concept seems to be reminiscent of a control system in which various outputs are computed in terms of the *same* state variables by using different "output functions."

3. THE COMPUTATION OF $D(X, Y)$

To compute $D(X, Y)$, we make use of the fact that this index has the recursive properties given above. The idea is essentially one of computing the distance $D(X_i, Y_j)$ between the prefixes of X and Y. The computation of the distances has to be done in a systematic manner, so that any quantity $D(X_i, Y_j)$ is computed before its value is required in any further computation. Just as in the case of the previous string edit algorithms [3–6, 10, 11, 13, 15, 18, 19, 21, 25], this can be actually done in a straightforward manner by tracing the underlying graph, commonly referred to as a *trellis*, and maintaining an array $Z(i,j)$ defined for all $0 \leq i \leq N$ and $0 \leq j \leq M$, when $|X| = N$ and $|Y| = M$. The quantity $Z(i,j)$ is nothing but $D(X_i, Y_j)$. We will discuss the properties of our particular trellis subsequently.

Initially, the weight associated with the origin $Z(0,0)$ is assigned the value zero, and the weights associated with the vertices on the axes are evaluated. Thus, $Z(i,0)$ and $Z(0,j)$ are computed using Lemmas 0a and 0b for all $1 \leq i \leq N$ and $1 \leq j \leq M$. The value for $Z(1,1)$ is then computed as a special computation outside any loop. Subsequently, the values for the lines $i = 1$ and $j = 1$ are traversed, and the distances associated with the vertices on these lines are computed using the previously computed values and Lemmas 0d and 0e. Finally, the weights corresponding to strict "interior" values (i.e., whenever i, $j > 1$) of the variables are computed.

The algorithm to compute $Z(\cdot,\cdot)$ is given below.

ALGORITHM GeneralizedDistance

Input: The strings $X = x_1 \dots x_N$ and $Y = y_1 \dots y_M$, and the set of elementary edit distances defined using the five elementary functions $d_s(\cdot,\cdot), d_i(\cdot), d_e(\cdot), d_{sq}(\cdot,\cdot)$, and $d_{ex}(\cdot,\cdot)$.

Output: The distance $D(X,Y)$ associated with transforming X to Y using the five edit operations of substitution, insertion, deletion, squashing, and expansion.

Method:

```
    $Z(0,0) \leftarrow 0$
    For $i \leftarrow 1$ to $N$ Do
        $Z(i,0) \leftarrow Z(i-1,0) + d_e(x_i)$
    For $j \leftarrow 1$ to $M$ Do
        $Z(0,j) \leftarrow Z(0,j-1) + d_i(y_j)$
    $Z(1,1) \leftarrow \text{Min}[Z(0,1) + d_e(x_1), Z(1,0) + d_i(y_1),$
                $Z(0,0) + d_s(x_1, y_1)]$
    For $i \leftarrow 2$ to $N$ Do
        $Z(i,1) \leftarrow \text{Min}[Z(i-1,1) + d_e(x_i), Z(i,0) + d_i(y_1),$
                $Z(i-1,0) + d_s(x_i, y_1),$
                $Z(i-2,0) + d_{sq}(x_{i-1}x_i, y_1)]$
    For $j \leftarrow 2$ to $M$ Do
        $Z(1,j) \leftarrow \text{Min}[Z(0,j) + d_e(x_1), Z(1,j-1) + d_i(y_j),$
                $Z(0,j-1) + d_s(x_1, y_j),$
```

```
                    $Z(0, j-2) + d_{ex}(x_1, y_{j-1}y_j)]$
    **For** $i \leftarrow 2$ to $N$ **Do**
        **For**$j \leftarrow 2$ to $M$ **Do**
            $Z(i,j) \leftarrow \mathrm{Min}[Z(i-1,j) + d_e(x_i), Z(i,j-1) + d_i(y_j),$
                        $Z(i-1,j-1) + d_s(x_i, y_j),$
                        $Z(i-2,j-1) + d_{sq}(x_{i-1}x_i, y_j),$
                        $Z(i-1,j-2) + d_{ex}(x_i, y_{j-1}y_j)]$
    $D(X,Y) \leftarrow Z(N,M)$
**END** ALGORITHM **GeneralizeDistance**
```

The computational complexity of algorithms involving the string comparison is conveniently given by the number of symbol comparisons required by the algorithm [1, 20]. In this case, the number of symbol comparisons (or more relevantly, the number of invocations of the functions $d_e(\cdot)$, $d_i(\cdot)$, $d_s(\cdot,\cdot)\, d_{sq}(\cdot,\cdot)$, $d_{ex}(\cdot,\cdot)$). required by ALGORITHM GeneralizedDistance is clearly quadratic. Note that in the interior of the main loop, we will need at most five additions and the computation of the minimum of *a* fixed (at most five) quantities.

The lower bound result claimed in [29] naturally implies that our algorithm is optimal for the infinite-alphabet case. This is because, first of all, we have not placed any restrictions on the edit costs. Also, the lower bound of [29] applies to the more restricted problem of finding a minimum cost alignment. Finally, when squashing and expansions have infinite costs, our underlying problem contains the traditional string alignment problem as a special case.

3.1. *GRAPHICAL REPRESENTATION OF THE ALGORITHM*

As mentioned earlier, in the computation of various string similarity and dissimilarity measures, the underlying graph that has to be traversed is commonly called a *trellis* (or grid graph). This trellis is two-dimensional in the case of the GLD [2, 6, 13, 15, 18, 19], the length of the LCS [3–6, 12, 18], and the length of the shortest common supersequence [12] of *two* strings. Indeed, the same trellis can be traversed using various set operators to yield the set of the LCS's and the set of the shortest common supersequences [6]. The trellis becomes essentially three-dimensional when one has to compute string probabilities [10], constrained edit distances [25], and correct noisy subsequences [21, 22]. Although the trellis itself is two-dimensional in the former examples, because the graphs are *cycle-free* they can be represented and traversed by merely maintaining single-dimensional structures [4]. Similarly, space optimizations are possible in the case of computing string probabilities and constrained edit distances.

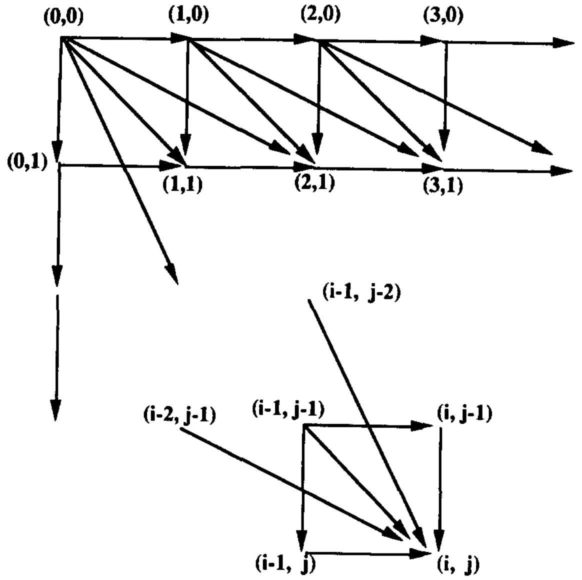

Fig. 1. The Trellis that has to be traversed in order to compute $D(X, Y)$. Note that the only edges terminating at (i, j) are those starting at $(i-2, j-1)$, $(i-1, j-1)$, $(i, j-1)$, $(i-1, j)$ and $(i-1, j-2)$.

Even though the set of edit operations has been expanded, the fundamental properties of the underlying graph remain the same. In this case, the vertices of the graph are the pairs $\langle i, j\rangle$, where $0 \le i \le N$, $0 \le j \le M$. The edges from a valid node $\langle i, j\rangle$ are directed arcs from $\langle i, j\rangle$ to $\langle i+1, j\rangle, \langle i, j+1\rangle, \langle i+1, j+1\rangle, \langle i+2, j+1\rangle$, and $\langle i+1, j+2\rangle$, wherever the target nodes are feasible. The graph essentially has arcs whenever a single edit operation can be applied. Indeed, the algorithm describes an efficient quadratic time scheme by which the trellis can be traversed.

For the sake of clarity, a pictorial representation of the graph is given in Figure 1.

3.2. COMPUTING THE BEST EDIT SEQUENCE

Just as in all the edit processes studied in the literature [3–6, 12, 14, 15, 18, 19], the traversal of the trellis not only yields the information about the distance between the strings X and Y. By virtue of the way the trellis has been traversed, the distances between the prefixes of the strings has also been maintained in the process of computation, and thus, the array Z contains information which can be used to compute the best edit alignment which yields the optimal edit distance. This is done by backtracking through the trellis from the array element (N, M) in the reverse direction of the arrows so as to reach the origin, always remembering the path that was used to reach the node which is currently being visited. Thus the actual sequence of edit operations can be printed out in the reverse order. Without further comment, we now present ALGORITHM Produce EditOperations, which has as its input the array $Z(\cdot,\cdot)$. To simplify the backtracking, we exclude the possibility of encountering negative values of i and j by rendering $Z(\cdot,\cdot)$ infinite whenever any index is negative.

ALGORITHM ProduceEditOperations

Input: The strings $X = x_1 \dots x_N$ and $Y = y_1 \dots y_M$, the set of elementary edit distances defined as in Algorithm Generalized Distance, and the array Z.

Output: The best edit alignment that can transform X to Y using the edit operations of substitution, insertion, deletion, squashing, and expansion.

Method:

```
Define $Z(i,j) \leftarrow \infty$ whenever $i < 0$ or $j < 0$.
$i \leftarrow N$
$j \leftarrow M$
While $(i \neq 0$ or $j \neq 0)$ Do
    If $(Z(i,j) = Z(i-1, j-1) + d_s(x_i, y_j))$ Then
        Print ("Substitute" $x_i$ "by" $y_j$)
        $i \leftarrow i - 1$
        $j \leftarrow j - 1$
    Else
      If $(Z(i,j) = Z(i, j-1) + d_i(y_j))$ Then
          Print("Insert" $y_j$)
          $j \leftarrow j - 1$
      Else
        If $(Z(i,j) = Z(i-1, j) + d_e(x_i))$ Then
            Print("Delete" $x_i$)
            $i \leftarrow i - 1$
        Else
            If $(Z(i,j) = Z(i-2, j-1)$
                        $+ d_{sq}(x_{i-1}x_i, y_j))$ Then
                Print("Squash" $x_{i-1}x_i$ "into" $y_i$)
                $i \leftarrow i - 2$
                $j \leftarrow j - 1$
            Else
```

```
                        If ($Z(i,j) = Z(i-1, j-2)$
                                 $+ d_{ex}(x_i, y_{j-1}y_j)$) Then
                            Print("Expand" $x_i$ "into" $y_{j-1}y_j$)
                            $i \leftarrow i - 1$
                            $j \leftarrow j - 2$
                        EndIf
                    EndIf
                EndIf
            EndIf
        EndIf
    EndWhile
END ALGORITHM ProduceEditOperations
```

A recursive version of the above which yields the edit sequence in the correct order can be easily written. A skeletal form of this procedure would be as follows:

ALGORITHM RecursiveProduceEditOperations (i, j)

Input: The strings $X = x_1 \dots x_N$ and $Y = y_1 \dots y_M$, the set of elementary edit distances defined as in Algorithm Generalized Distance, the array Z, and the indices i and j.
Output: The best edit alignment that can transform X_i to Y_j.
Method:

```
    If ($Z(i,j) = Z(i-1, j-1) + d_s(x_i, y_j)$) Then
        RecursiveProduceEditOperations ($i-1, j-1$)
        Print ("Substitute" $x_i$ "by" $y_j$)
    EndIf

END ALGORITHM RecursiveProduceEditOperations
```

Throughout this paper, we have only considered the case when the types of expansion and squashing errors involve transformations from a single character in one string to two characters in the second. It is easy to visualize the generalization of this when the number of characters involved in the squash/expand operations is a constant K, where $K \geq 2$. The resulting trellis then would have to be traversed in essentially the same way, except that at every internal node, the minimization would involve the computation of $2K + 1$ quantities. For example, if K is 3, the corresponding minimization in the interior of the trellis would involve the following expression:

$$\begin{aligned} Z(i,j) \leftarrow \mathbf{Min}\,[Z(i-1,j-1) + d_s(x_i,y_j), Z(i,j-1) + d_i(y_j),& \\ Z(i-1,j) + d_e(x_i),& \\ Z(i-2,j-1) + d_{sq}(x_{i-1}x_i,y_j), Z(i-1,j-2)& \\ + d_{ex}(x_i,y_{j-1}y_j),& \\ Z(i-3,j-1) + d_{sq}(x_{i-2}x_{i-1}x_i,y_j), Z(i-1,j-3)& \\ + d_{ex}(x_i,y_{j-2}y_{j-1}y_j)].& \end{aligned}$$

The algorithm would still be quadratic in the lengths of the strings as long as K is independent of M and N, which is not an unreasonable assumption, especially as the types of errors that are caused in a channel are typically not functions of the strings transmitted themselves.

The GLD, defined in terms of the standard edit operations, has been used to perform the automatic correction of noisy strings [2, 9, 16, 18], substrings [8], and subsequences [21, 22]. We believe that the distance defined using the expanded set operations can be used to perform analogous correction when the errors include the "bouncing" and "coalescing" of characters and phonemes. These concepts can also be applied to the comparison of molecular sequences when a single amino acid can be decomposed as a sequence of two (or more) compounds, each of which is represented by a single symbol.

4. CONCLUSIONS

Let X and Y be any two strings of finite length. The problem of transforming X to Y using the edit operations of substitution, deletion, and insertion has been extensively studied in the literature [1, 2, 6–11, 13, 15, 16, 18–21]. In this paper, we have considered the problem of editing X to Y when the set of edit operations is extended to include the squashing and expansion operations. In the squashing operation two (or more) contiguous characters of X can be transformed into a single character of Y, and in the expansion operation a single character in X may be expanded into two or more contiguous characters of Y. The case when the number of operations involved in the squash/expansion is two has been thoroughly analyzed, and the case when this number is larger than two has been alluded to. A quadratic time solution to the problem has been presented. This solution is optimal for the infinite-alphabet case.

I would like to thank John Andrusek and William Lee for their help in preparing the manuscript. I am especially grateful to an anonymous referee *who provided me with various comments regarding complexity issues and the comparison of these results with existing results.*

REFERENCES

1. A. V. Aho, D. S. Hirschberg, and J. D. Ullman, Bounds on the complexity of the longest common subsequence problem, *J. Assoc. Comput. Mach.* 23:1–12 (1976).
2. P. A. V. Hall and G. R. Dowling, Approximate string matching, *Comput. Surveys* 12:381–402 (1980).
3. D. S. Hirschberg, Algorithms for longest common subsequence problem, *J. Assoc. Comput. Mach.* 24:664–675 (1977).
4. D. S. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Commun. Assoc. Comput. Mach.* 18:341–343 (1975).
5. J. W. Hunt and T. G. Szymanski, A fast algorithm for computing longest common subsequences, *Commun. Assoc. Comput. Mach.* 20:350–353 (1977).
6. 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).
7. R. L. Kashyap and B. J. Oommen, Similarity measures for sets of strings, *Internat. J. Comput. Math.* 13:95–104 (1983).
8. R. L. Kashyap and B. J. Oommen, The noisy substring matching problem, *IEEE Trans. Software Engng.* SE-9:365–370 (1983).
9. 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).
10. R. L. Kashyap and B. J. Oommen, String correction using probabilistic methods, *Pattern Recog. Lett.* 147–154 (1984).
11. A. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, *Sov. Phys. Dokl.* 10:707–710 (1966).
12. D. Maier, The complexity of some problems on subsequences and supersequences, *J. Assoc. Comput. Mach.* 25:322–336 (1978).
13. W. J. Masek and M. S. Paterson, A faster algorithm computing string edit distances, *J. Comput. System Sci.* 20:18–31 (1980).
14. S. B. Needleman and C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J. Mol. Biol.* 443–453 (1970).
15. T. Okuda, E. Tanaka, and T. Kasai, A method of correction of garbled words based on the Levenshtein metric, *IEEE Trans. Comput.* C-25:172–177 (1976).
16. J. L. Peterson, Computer programs for detecting and correcting spelling errors, *Commun. Assoc. Comput. Mach.* 23:676–687 (1980).
17. D. Sankoff, Matching sequences under deletion/insertion constraints, *Proc. Nat. Acad. Sci. U.S.A.* 69:4–6 (1972).
18. D. Sankoff and J. B. Kruskal, *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison-Wesley, MA, 1983.
19. R. A. Wagner and M. J. Fischer, The string to string correction problem, *J. Assoc. Comput. Mach.* 21:168–173 (1974).
20. C. K. Wong and A. K. Chandra, Bounds for the string editing problem, *J. Assoc. Comput. Mach.* 23:13–16 (1976).
21. B. J. Oommen, Recognition of noisy subsequences using constrained edit distances, *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-9:676–685 (1987).

22. B. J. Oommen and E. T. Floyd, An improved algorithm for the recognition of noisy subsequences, *Proceedings of the 1991 IASTED International Symposium on Artificial Intelligence Applications and Neural Networks*, Zurich, 1991, pp. 145–147.
23. J. Golic and M. Mihaljevic, A noisy clock-controlled shift register cryptanalysis concept based on sequence comparison approach, *Proceedings of EUROCRYPT 90*, Aarhus, Denmark, 1990, pp. 487–491.
24. R. Lowrance and R. A. Wagner, An extension of the string to string correction problem, *J. Assoc. Comput. Mach.* 22:177–183 (1975).
25. B. J. Oommen, Constrained string editing, *Inform. Sci.* 40:267–284 (1987).
26. K. Abe and N. Sugita, Distances between strings of symbols — Review and remarks, *Proceedings of the Sixth International Conference on Pattern Recognition*, (1982), pp. 172–174.
27. E. Ukkonen, Algorithms for approximate string matching, *Inf. Contr.* 64:100–118 (1985).
28. R. A. Wagner, On the complexity of the extended string-to-string correction problem, *Proceedings of the Seventh Symposium on the Theory of Computing*, 1975, pp. 218–223.
29. X. Huang, A lower bound for the edit distance problem under an arbitrary cost function, *Inf. Proc. Lett.* 27:319–321 (1988).

Received 2 January 1993; revised 25 February 1994