ADVANCES IN APPLIED MATHEMATICS 6, 129–134 (1985)

Dynamic Programming Algorithms for Picture Comparison

MICHAEL S. WATERMAN*

Departments of Mathematics and Biological Sciences, University of Southern California, Los Angeles, California 90089-1113

Two-dimensional arrays can be compared by a generalization of dynamic programming algorithms for string comparison. Earlier algorithms have computational complexity $O(N^6)$ for comparison of two $N \times N$ arrays. The computational complexity is reduced to $O(N^4)$ in general and $O(N^2)$ algorithms are pointed out for the range limited case. An example is given to illustrate the lack of knowledge of mathematical properties of these algorithms. The problem of finding an algorithm to compute the minimum number of insertions, deletions, and substitutions to transform one array into another remains open.

1. INTRODUCTION

A dynamic programming algorithm for measuring distance between two strings was first proposed by Levenshtein [4] and has been rediscovered by several authors. Applications of the algorithm have been varied and include error-correcting codes [12], spelling correction [14], geological stratigraphy [10], comparison of DNA and protein sequences [7, 9], speech recognition [6], birdsong studies [1], and handwriting recognition [2]. A recent book edited by Sankoff and Kruskal [8] surveys the theory and practice of dynamic programming sequence comparison.

It would be extremely useful to have a generalization of the one-dimensional Levenshtein (1DL) algorithm to allow comparison of two-dimensional arrays. Applications could include the handwriting recognition problem referred to above, for example. Two groups, Moore [5] in England and Tanaka and Kikuchi [13] in Japan, have independently proposed such an algorithm, which is referred to here as the 2DL algorithm.

In this note, the computational complexity of the 2DL algorithm is reduced from $O(N^6)$ to $O(N^4)$ for comparison of two $N \times N$ arrays. This

*This work was supported by a grant from System Development Foundation.

0196-8858/85 $7.50

reduction makes it practical to compare small arrays of interest. In addition, the mathematical properties of the 2DL algorithm are briefly considered. The algorithms proposed so far do not compute a metric, and work remains to be done on these interesting problems.

2. String Comparison

It is useful to recall the 1DL metric for comparing two strings $\mathbf{x} = x_1x_2 \cdots x_n$ and $\mathbf{y} = y_1y_2 \cdots y_m$ over some alphabet. The algorithm gives a minimum cost sequence for changing the string x into the string y. The cost of changing one letter into another, a substitution, is ν, and the cost of inserting or deleting a letter is δ. More general cost functions are possible but this simple one will satisfy our purposes. The distance between x and y is defined by

$$d(\mathbf{x},\mathbf{y}) = \min\{k\delta + l\nu\}$$

where the minimum is over all possible ways of changing x into y, l is the number of substitutions, and k is the number of deletions and insertions. If each letter in the x sequence is displayed above the letter in the y sequence it is mapped into, we obtain a display of $\mathbf{x} \to \mathbf{y}$ or an alignment of x and y. For example woork and word might align by

$$\begin{array}{c} \mathrm{w\,o\,o\,r\,k} \\ \mathrm{w\,-\,o\,r\,d} \end{array}$$

where — is inserted into "word" to indicate a deletion of the first o in woork. The alignment has one deletion and one substitution so its cost is $\delta + \nu$.

To understand the dynamic programming algorithm for computing $d(\mathbf{x},\mathbf{y})$ set

$$d_{ij} = d(x_1x_2 \cdots x_i, y_1y_2 \cdots y_j).$$

Consider the various ways an alignment of $x_1x_2 \cdots x_i$ and $y_1y_2 \cdots y_j$ can end. The top line can end with x_i or — and the bottom line can end with y_j or —. Ending both with — is entirely uninteresting as it specifies the fate of either x_i or y_j. Therefore the alignment can end in $2^2 - 1 = 3$ ways:

$$\begin{array}{c} x_i \\ - \end{array} \quad \text{or} \quad \begin{array}{c} x_i \\ y_j \end{array} \quad \text{or} \quad \begin{array}{c} - \\ y_j \end{array}.$$

At least one of these belongs to an alignment with cost d_{ij} so

$$d_{ij} = \min\{d_{i-1,j} + \delta, d_{i-1,j-1} + \nu I, d_{i,j-1} + \delta\},$$

where $I = 0$ if $x_i = y_j$ and $I = 1$ if $x_i \neq y_j$. This algorithm uses time $O(N^2)$.

3. Picture Comparison

Moore [5] and Tanaka and Kikuchi [13] have independently generalized the 1DL algorithm to two-dimensional picture comparison. Let $\mathbf{x} = \{x(i, j): 1 \le i, j \le N\}$, and $\mathbf{y} = \{y(i, j): 1 \le i, j \le N\}$ be two-dimensional arrays. Moore writes an algorithm for comparing x and y while Tanaka and Kikuchi define 2DL distance to be the minimum cost of transforming x into y by deletions, insertions, and substitutions. Tanaka and Kikuchi go on to write an algorithm which they justify by relating it to an algorithm which is essentially that which Moore presents. We first study the algorithm (as presented by Moore), which we call the 2DL algorithm for reasons that will become clear. Then we will comment on implications of such an algorithm.

First let $\mathbf{x}(\bar{i}, \bar{j}) = \{x(k, l): 1 \le k \le i, 1 \le l \le j\}$ and $\mathbf{y}(\bar{m}, \bar{n}) = \{y(k, l): 1 \le k \le m, 1 \le l \le n\}$. If we think of aligning rows (columns) of x with rows (columns) of y in the same way as substituting an "x" row (column) with a "y" row (column), then there are several ways of ending an $x(\bar{i}, \bar{j})/y(\bar{m}, \bar{n})$ alignment. The lower right edges of the alignment correspond to the right-hand letters for the case of alignment of two linear sequences. For further notation, following Tanaka and Kikuchi, let $x(i, \bar{j})$ denote $x(i, l)$, $1 \le l \le j$. We see that the edges can be composed of the various ways of matching and/or deleting four elements: $x(i, \bar{j})$ and $y(m, \bar{n})$, the two rows, and $x(\bar{i}, j)$ and $y(\bar{m}, n)$, the two columns. The only configuration that does not make sense is, as above, the one that involves matching two row deletions and two column deletions. Therefore an algorithm, which we call the 2DL algorithm, analogous to the 1DL algorithm will involve minimizing $2^4 - 1 = 15$ terms.

Let the 2DL cost between $x(\bar{i}, \bar{j})$ and $y(\bar{n}, \bar{m})$ be $D(i, j; n, m)$. By the analogy presented above this term is the minimum of

(1) $D(i-1, j; m, n) + d(x(i, \bar{j}), —)$
(2) $D(i, j-1; m, n) + d(x(\bar{i}, j), —)$
(3) $D(i, j; m-1, n) + d(y(m, \bar{n}), —)$
(4) $D(i, j; m, n-1) + d(y(\bar{m}, n), —)$
(5) $D(i-1, j-1; m, n) + d(x(i, \bar{j}), —) + d(x(\overline{i-1}, j), —)$
(6) $D(i, j; m-1, n-1) + d(y(n, \bar{m}), —) + d(y(\overline{n-1}, m), —)$
(7) $D(i-1, j; m-1, n) + d(x(i, \bar{j}), y(m, \bar{n}))$
(8) $D(i-1, j; m, n-1) + d(x(i, \bar{j}), —) + d(y(\bar{m}, n), —)$
(9) $D(i, j-1; m-1, n) + d(x(\bar{i}, j), —) + d(y(m, \bar{n}), —)$
(10) $D(i, j-1; m, n-1) + d(x(\bar{i}, j), y(\bar{m}, n))$
(11) $D(i-1, j-1; m-1, n) + d(x(i, \bar{j}), y(m, \bar{n})) + d(x(\overline{i-1}, j), —)$
(12) $D(i-1, j-1; m, n-1) + d(x(i, \overline{j-1}), —) +$

$d(x(\bar{i}, j), y(\bar{m}, n))$

(13) $D(i-1, j; m-1, n-1) + d(x(i, \bar{j}), y(m, \bar{n}))$
$+ d(y(\overline{m-1}, n), —)$

(14) $D(i, j-1; m-1, n-1) + d(x(\bar{i}, j), y(\bar{m}, n))$
$+ d(y(m, \overline{n-1}), —)$

(15) $D(i-1, j-1; m-1, n-1) + \min\{d(x(\bar{i}, j), y(\bar{m}, n)) +$
$d(x(i, \overline{j-1}), y(m, \overline{n-1})), d(x(\overline{i-1}, j), y(\overline{m-1}, n))$
$+ d(x(i, \bar{j}), y(m, \bar{n}))\}.$

The computational complexity of this algorithm, for two $N \times N$ arrays, is

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{m=1}^{N} \sum_{n=1}^{N} (4im + 4jn + C) = O(N^6).$$

The cost from terms (11) through (15) raises the computation to N^6, and each term comes from the comparison of two linear strings. Define

$$d_1(i, j; m, n) = d(x(\bar{i}, j), y(\bar{m}, n)),$$

and

$$d_2(i, j; m, n) = d(x(i, \bar{j}), y(m, \bar{n})).$$

From our discussion of string comparison above, we see that

$$\begin{aligned} d_1(i, j; m, n) = \min\{ & d_1(i-1, j; m, n) + \delta, \\ & d_1(i-1, j; m-1, n) + \nu, \\ & d_1(i, j; m-1, n) + \delta\} \end{aligned}$$

and

$$\begin{aligned} d_2(i, j; m, n) = \min\{ & d_2(i, j-1; m, n) + \delta, \\ & d_2(i, j-1; m, n-1) + \nu, \\ & d_2(i, j; m, n-1) + \delta\}. \end{aligned}$$

This simple device allows recursive computation of the string comparisons and the computational complexity of the improved 2DL algorithm is

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{m=1}^{N} \sum_{m=1}^{N} (C) = O(N^4).$$

a a a b c	*a a a b*
a a a c b	*a a a b*
a a a b c	*a a a b*
b c b c c	*b b b b*
c b c c b	
x	y

FIG. 1. Two arrays for comparison.

It is clear that storage $O(N^2)$ is required to compute $D(N, N; N, N)$ while storage $O(N^4)$ is required to produce the correspondence between the arrays.

4. DISCUSSION

The algorithm requires that, if row or column deletions are not optimal, either bottom row or right columns are matched. This allowed Tanaka and Kikuchi to present an algorithm with six terms to minimize, as opposed to fifteen above. Their algorithm runs in $O(N^6)$ steps and, as above, can be reduced to $O(N^4)$. Range limited comparison of two arrays could be accomplished in $O(N^2)$ steps. Instead of pursuing these further economies, we make some remarks about the properties of 2DL distance.

The Tanaka and Kikuchi definition of distance is the minimum cost of insertions, deletions, and substitutions to change array x into array y. This desirable distance is not computed by our 2DL distance as can be seen by the example presented in Fig. 1. Let insertions and deletions cost 1 and substitutions cost 1.5. It is clear that y can be obtained from x by deletions of all letters equal to C, for a cost of 9. However, $D(\mathbf{x}, \mathbf{y}) = 12$. Our 2DL distance does not find array matchings which ends in a configuration which "weaves" outer rows and columns in this manner. Currently, no known algorithm computes the minimum number of insertions, deletions, and substitutions to map x into y.

To summarize, computation of 2DL distance for two $N \times N$ arrays can be accomplished in $O(N^4)$ and, in the range limited case, $O(N^2)$. These efficiencies make this distance practical for some problems. If the most common regions between two arrays are desired, corresponding modification of an algorithm of Smith and Waterman [11] is, with some careful effort, possible. Several related algorithms have been presented in Darling and Waterman [3] which studies the probability distribution of the volume of the largest matching square and rectangle between random d-dimensional arrays. The example of Fig. 1 raises several questions, however, as to the mathematical properties of 2DL distance.

REFERENCES

1. E. A. ARMSTRONG, "A Study of Bird Song," 2nd ed., p. 206, Dover, New York, 1973.
2. D. J. BURR, Designing a handwriting reader, *IEEE Trans. Pattern Anal. Machine Intelligence* **PAMI-5** (1983), 554–559.
3. R. W. R. DARLING AND M. S. WATERMAN, Success runs in d-dimensions: Algorithms and laws of large numbers, *Adv. in Math.* **55** (1985).
4. V. I. LEVENSHTEIN, Binary codes with correction of deletions, insertions and substitutions of symbols, *Dokl. Akad. Nauk. SSSR* **163** (1963), 845–848.
5. R. K. MOORE, A dynamic programming algorithm for the distance between two finite areas, *IEEE Trans. Pattern Anal. Machine Intelligence* **PAMI-1, 1** (1979), 86–88.
6. H. SAKOE AND S. CHIBA, Dynamic-programming algorithm optimization for spoken word recognition, *IEEE Trans. Acoust. Speech Signal Process.* **ASSP-26** (1978), 43–49.
7. D. SANKOFF, Matching sequences under deletion/insertion constraints, *Proc. Nat. Acad. Sci. U.S.A.* **69**, No. 1 (1972), 4–6.
8. D. SANKOFF AND J. B. KRUSKAL (Eds.), "Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison," Addison-Wesley, London, 1983.
9. P. H. SELLERS, On the theory and computation of evolutionary distances, *SIAM J. Appl. Math.* **26** (1974), 787–793.
10. T. F. SMITH AND M. S. WATERMAN, New stratigraphic correlation techniques, *J. Geol.* **88** (1980), 451–457.
11. T. F. SMITH AND M. S. WATERMAN, Identification of common molecular subseqences, *J. Molecular Bio.* **147** (1981), 195–197.
12. E. TANAKA AND T. KASAI, Synchronization and substitution of error-correcting codes for the Levenshtein metric, *IEEE Trans. Inform. Theory* **IT-22, 2** (1976), 156–162.
13. E. TANAKA AND Y. KIKUCHI, A metric between pictures, *Systems-Comput. Controls* **11**, No. 6 (1980), 49–57.
14. R. A. Wagner and M. J. Fisher, The string to string correction problem, *J. Assoc. Comput. Mach.* **21** (1974), 168–173.