アルゴリズム 44－5
（1995. 3. 17）

二次元配列間の距離について

阿久津 達也
群馬大学 工学部 情報工学科

二次元配列の近似マッチング問題として従来提案されてきたものは縦方向と横方向の扱いが異なる不自然なものであった。本稿では、より自然な近似マッチングを行うために、縦方向と横方向が対等に考慮される二次元配列間の距離 (誤差) を定義する。そして、MAX-2SAT 問題を還元することにより、一般的にはこの距離を計算することは NP 困難であることを示す。一方、特殊な、しかし、ある程度現実的な場合には最短経路問題に帰着させることにより多項式時間で計算できることも示す。

On the Editing Distance between Two-Dimensional Arrays

Tatsuya Akutsu
Department of Computer Science, Gunma University
1-5-1 Tenjin, Kiryu, Gunma 376 Japan
e-mail: akutsu@cs.gunma-u.ac.jp

This paper proposes an editing distance problem between two-dimensional arrays, in which rows are treated in the same way as in columns. This problem is important for approximate pattern matching between two-dimensional arrays. The problem is proved to be NP-hard by a reduction from MAX-2SAT. However a polynomial time algorithm is shown for a special but practical case, in which the problem is reduced to the shortest path problem.

1 Introduction

Recently pattern matching problems for two-dimensional arrays have been studied extensively. Two-dimensional pattern matching problems are important for handling two-dimensional images. For example, they are useful for image recognition and data compression. In such practical cases, approximate matching seems much more important than exact matching since errors are inevitable in most cases. However a few studies have been done for approximate matching [1, 2, 4, 6]. Moreover errors considered in [1, 2, 4] are restricted as follows: a deletion or insertion affects only the column it appears in. This restriction does not seem to be natural because treatment of rows is different from that of columns. Motivated by the above facts, we consider an editing distance problem between two-dimensional arrays in which rows are treated in the same way as in columns. Note that most approximate matching problems are defined using editing distances (or, equivalently differences) [5, 7, 8]. In this paper, we show that the editing distance problem is NP-hard in general, while we present a polynomial time algorithm for a special but practical case.

Here we briefly review previous results. Approximate matching of strings has been well studied and several efficient algorithms have been developed [5]. Various editing distances have been proposed for approximate tree matching [7, 8]. Krithivasan and Sitalakshmi studied approximate matching of two-dimensional arrays [4], while their results were improved by Amir and Landau [1]. Amir and Farach studied approximate matching of non-rectangular figures [2]. However errors considered in these three studies are restricted as mentioned above. Recently Landau and Vishkin proposed a new approach to pattern matching of images [6]. Although it seems a reasonable approach towards practical pattern matching, it is not robust against errors that affect far positions.

2 Definition of the Editing Distance

In this section, we define an editing distance between two-dimensional arrays. Let $A[1..m, 1..n]$ be an $m \times n$ two-dimensional array over an alphabet Σ. Let $D = \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$ be the set of directions. Then *editing operations* are defined as below (see also Fig. 1), where we consider only the right direction ($\rightarrow$). Deletions and insertions for the other directions are defined in a similar way.

Substitution: ($sub(i, j, x)$) $A[i, j]$ is replaced by $x \in \Sigma$ (i.e., $A[i, j] := x$).

Insertion: ($ins(i, j, x, dir)$) $x \in \Sigma$ is inserted at position (i, j), and the contents of $A[i, j..(n-1)]$ are shifted to the right direction, where $A[i, j..k]$ denotes the subarray that consists of elements of row i and columns j to k ($j \leq k$). The content of $A[i, n]$ is ignored.

Deletion: ($del(i, j, x, dir)$) The content of $A[i, j]$ is deleted, and the contents of $A[i, 1..(j-1)]$ are shifted to the right direction. $A[i, 1]$ is replaced by $x \in \Sigma$.

Let C_S be the cost per substitution, C_I be the cost per insertion, and C_D be the cost per deletion, where they satisfy the triangular inequality: $C_S \leq C_I + C_D$. Let $E = (e_1, e_2, \cdots, e_k)$ be a sequence of editing operations, which transforms $A[1..m, 1..n]$ to $B[1..m, 1..n]$. Then the *cost* of E is defined to be the sum of the costs of $e_1, \cdots, e_k$. The *editing distance* from $A[1..m, 1..n]$ to $B[1..m, 1..n]$ is defined by the minimum cost of the editing sequence which transforms $A[1..m, 1..n]$ to $B[1..m, 1..n]$.

You may think that the definition of a deletion operation is unusual. However we employed the above definition because we wanted to make a deletion symmetric to an insertion. Indeed, $del(i, j, \leftarrow, A[i, n])$ transforms $B[1..m, 1..n]$ to $A[1..m, 1..n]$ if $ins(i, j, \rightarrow, B[i, j])$ transforms

$A[1..m, 1..n]$ to $B[1..m, 1..n]$. Note that the results in this paper do not change even if we use the following definition of a deletion operation.

Deletion': ($del'(i, j, dir)$) The content of $A[i, j]$ is deleted, and the contents of $A[i, 1..(j-1)]$ are shifted to the right direction. $A[i, 1]$ is replaced by a special symbol #.

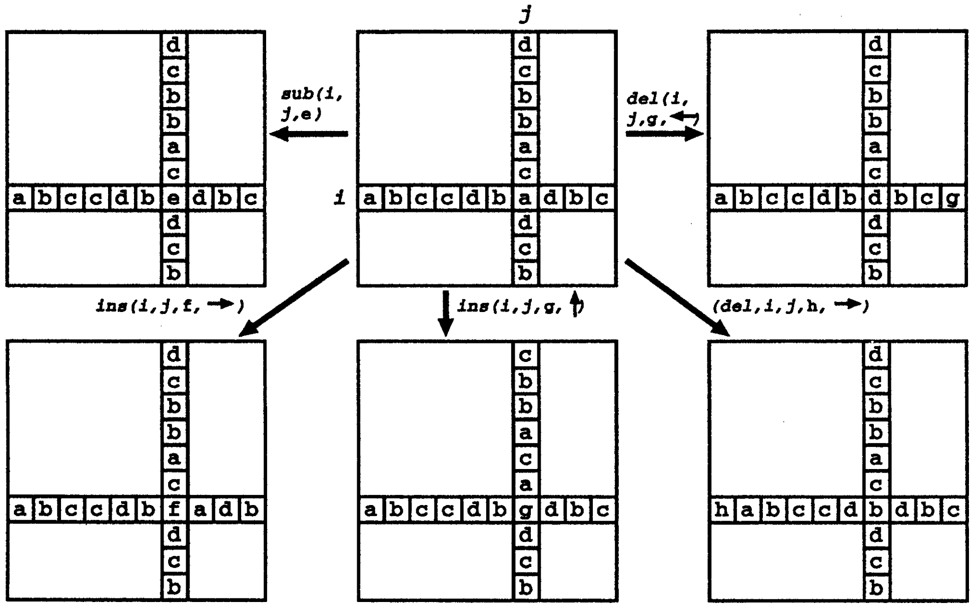

Figure 1: Editing operations.

3 NP-hardness Result

In this section, we show that computing the editing distance is NP-hard by means of a polynomial time reduction from MAX-2SAT.

It is well known that MAX-2SAT defined below is NP-complete [3].

Instance: A set $V = \{x_1, \cdots, x_N\}$ of variables, a collection $C = \{c_1, \cdots, c_M\}$ of clauses over V such that each clause consists of two literals, and a positive integer K.

Question: Is there a truth assignment for V that simultaneously satisfies at least K of the clauses in C ?

In the following, x_i denotes a positive literal, $\overline{x_i}$ denotes a negative literal, and $var(c_i)$ denotes the set of variables appearing in c_i, where we assume without loss of generality that $|var(c_i)| = 2$ for each clause c_i and $M > N$.

From an instance of MAX-2SAT, we construct arrays $A[1..m, 1..n]$ and $B[1..m, 1..n]$ over $\Sigma = \{T, F, 1, 2, \cdots, L, -1, -2, \cdots, -L, 0\}$ in the following way (see Fig. 2), where L is a constant such that $L \gg M$ (e.g., $L = 100M$).

Let H be a constant such that $H \gg LM$ and $(H \bmod 2L) = 0$ (e.g., $H = 100LM$). Let $m = (2M+1)H$ and $n = (N+1)H$. Here we define the following functions:

$$f_1(c_i,x_j) = \begin{cases} \boldsymbol{T}, & x_j \in c_i, \\ \boldsymbol{F}, & \overline{x_j} \in c_i, \\ 1, & \text{otherwise.} \end{cases} \qquad f_2(c_i,x_j) = \begin{cases} \boldsymbol{F}, & x_j \in c_i, \\ \boldsymbol{T}, & \overline{x_j} \in c_i, \\ 1, & \text{otherwise.} \end{cases}$$

$$id(i) = \begin{cases} i \bmod L, & 1 \le (i \bmod 2L) \le L, \\ -L, & (i \bmod 2L) = 0, \\ -(i \bmod L), & \text{otherwise.} \end{cases} \qquad \begin{aligned} I(i) &= \lfloor (i+H+L)/2H \rfloor. \\ J(j) &= \lfloor (j+L)/H \rfloor. \end{aligned}$$

Moreover we define the following notations (see Fig. 3):

$$\begin{array}{llll} posL(j) & \longleftrightarrow & j < NH \wedge (j \bmod H) = H - L + 1, & posR(j) \longleftrightarrow j > H \wedge (j \bmod H) = L + 1, \\ posU(i) & \longleftrightarrow & i < 2MH \wedge (i \bmod H) = H - L + 1, & posD(i) \longleftrightarrow i > H \wedge (i \bmod H) = L + 1, \\ posT(i) & \longleftrightarrow & (i \bmod 2H) = H + 1, & posC(j) \longleftrightarrow j > 1 \wedge (j \bmod H) = 1, \\ posB(i) & \longleftrightarrow & i > 1 \wedge (i \bmod 2H) = 1. & \end{array}$$

Then a part of $A[1..m, 1..n]$ is defined by $A[i,j] = \begin{cases} f_1(c_{I(i)}, x_{J(j)}), & posU(i) \wedge posC(j), \\ f_2(c_{I(i)}, x_{J(j)}), & posD(i) \wedge posC(j). \end{cases}$

The other part of $A[1..m, 1..n]$ is defined by $A[i,j] = \begin{cases} -id(i), & posC(j), \\ id(j), & \sim posC(j) \wedge (posT(i) \vee posB(i)), \\ 0, & \text{otherwise.} \end{cases}$

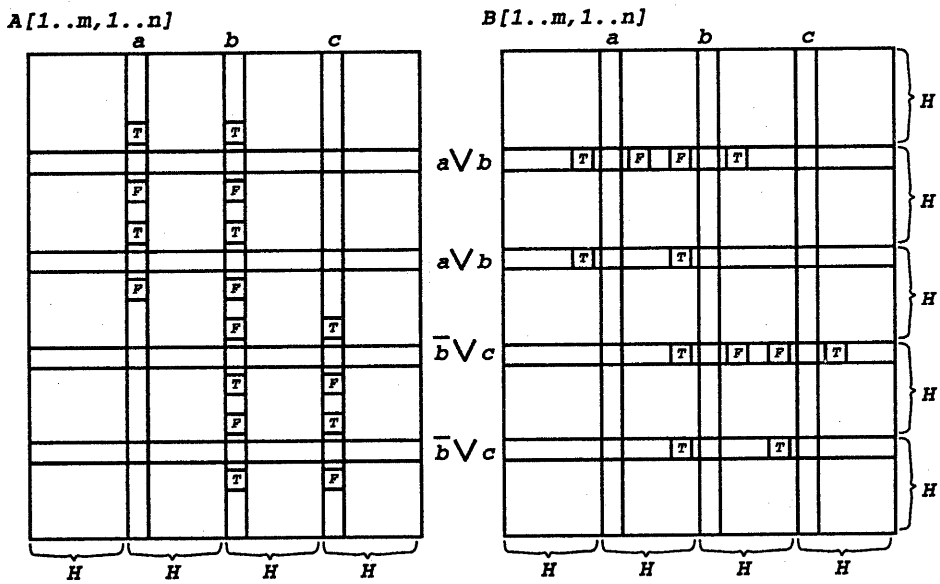

Figure 2: Arrays constructed from $\{a \vee b,\ \overline{b} \vee c\}$.

Next we describe the construction of $B[1..m, 1..n]$. We define three functions as below:

$$g_1(c_i,x_j) = \begin{cases} \boldsymbol{T}, & var(c_i) = \{x_j, x_k\} \wedge j < k, \\ \boldsymbol{F}, & var(c_i) = \{x_j, x_k\} \wedge j > k, \\ 1, & \text{otherwise.} \end{cases} \quad g_2(c_i,x_j) = \begin{cases} \boldsymbol{F}, & var(c_i) = \{x_j, x_k\} \wedge j < k, \\ \boldsymbol{T}, & var(c_i) = \{x_j, x_k\} \wedge j > k, \\ 1, & \text{otherwise.} \end{cases}$$

$$g_3(c_i,x_j) = \begin{cases} \boldsymbol{T}, & x_j \in var(c_i), \\ 1, & \text{otherwise.} \end{cases}$$

Then a part of $B[1..m, 1..n]$ is defined by $B[i,j] = \begin{cases} g_1(c_{I(i)}, x_{J(j)}), & posT(i) \wedge posL(j), \\ g_2(c_{I(i)}, x_{J(j)}), & posT(i) \wedge posR(j), \\ g_3(c_{I(i)}, x_{J(j)}), & posB(i) \wedge posL(j). \end{cases}$

The other part of $B[1..m, 1..n]$ is defined by $B[i,j] = \begin{cases} -id(j), & posT(i) \vee posB(i), \\ id(i), & \sim(posT(i) \vee posB(i)) \wedge posC(j), \\ 0, & \text{otherwise.} \end{cases}$

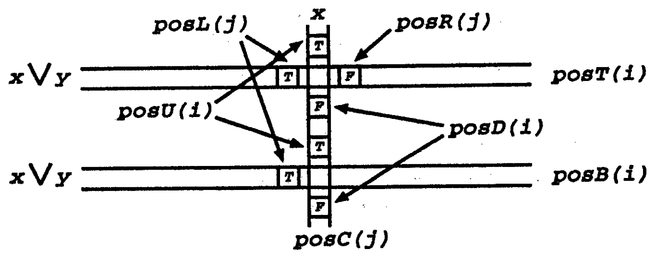

Figure 3: Positions specified by $posL(j), posR(j), \cdots$.

Lemma 1: Let $C_S = C_D = C_I = 1$. Then, there exists an editing sequence from $A[1..m, 1..n]$ to $B[1..m, 1..n]$ with cost at most $2LM+LN+9M-2K$ if and only if there exists an assignment that satisfies at least K clauses in C.

(Proof) In this paper, we only show that there exists an editing sequence with cost at most $2LM+LN+9M-2K$ if there exists an assignment that satisfies at least K clauses. Although the proof for the converse property is more complicated, it can be proved in a similar way.

From an assignment that satisfies K clauses, we construct an editing sequence E in the following way (see Fig. 4). First, we let $E := \{\}$. Next, for $j = 1$ to N,

$$(ins(1, jH+1, L, \downarrow), ins(1, jH+1, L-1, \downarrow), \cdots, ins(1, jH+1, 1, \downarrow))$$

is appended to E if x_j is assigned to T, otherwise

$$(ins((2M+1)H, jH+1, -1, \uparrow), ins((2M+1)H, jH+1, -2, \uparrow), \cdots, ins((2M+1)H, jH+1, -L, \uparrow))$$

is appended to E. Next, for $i = 1$ to M,

$$(ins((2i-1)H+1, (N+1)H, 1, \leftarrow), ins((2i-1)H+1, (N+1)H, 2, \leftarrow), \cdots, ins((2i-1)H+1, (N+1)H, L, \leftarrow))$$

is appended to E if c_i is satisfied by x_j where $var(c_i) = \{x_j, x_k\}$ and $j < k$, otherwise

$$(ins((2i-1)H+1, 1, -L, \rightarrow), ins((2i-1)H+1, 1, -(L-1), \rightarrow), \cdots, ins((2i-1)H+1, 1, -1, \rightarrow))$$

is appended to E. Next, for $i = 1$ to M,

$$(ins(2iH+1, (N+1)H, 1, \leftarrow), ins(2iH+1, (N+1)H, 2, \leftarrow), \cdots, ins(2iH+1, (N+1)H, L, \leftarrow))$$

is appended to E. Finally, substitutions for transforming to $B[1..m, 1..n]$ are appended to E.

Then it is easy to see that the total cost of insertions is $2LM + LN$. Thus we consider the total cost of substitutions. For each clause that is satisfied, the number of substitutions is $6+1=7$. For each clause that is not satisfied, the number of substitutions is $6+3=9$. Therefore the total cost is $2LM + LN + 6M + 3(M-K) + K = 2LM + LN + 9M - 2K$. □

Since the construction can be done in polynomial time, we obtain the following theorem.

Theorem 1: Computing the editing distance is NP-hard.

Although the size of an alphabet Σ is not bounded in the above reduction, it is possible to modify the reduction so that $|\Sigma| = 5$ holds. Note that deciding whether or not the editing distance is at most K is NP-complete because it is trivially in NP.

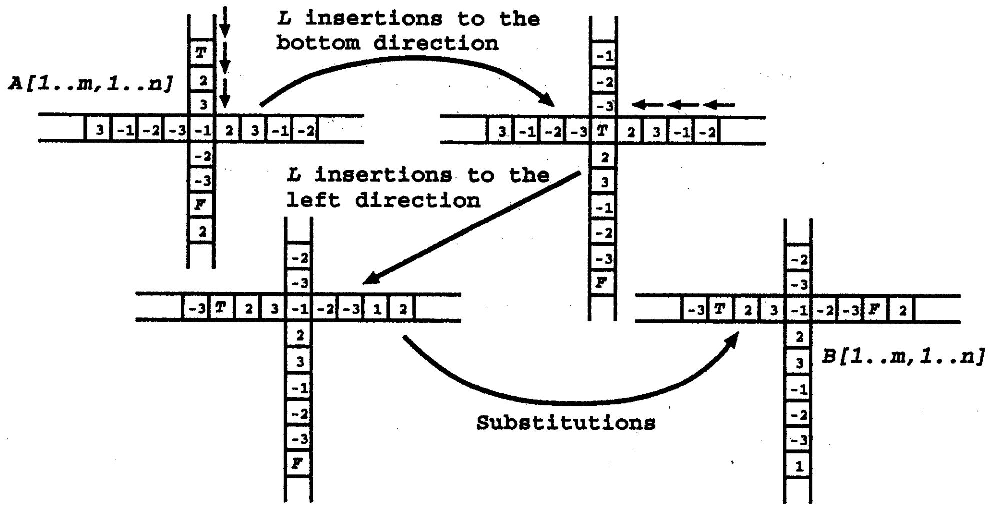

Figure 4: A part of an editing sequence constructed from an assignment.

4 A Polynomial Time Algorithm for a Special Case

Although a hardness result is shown, we may obtain a polynomial time algorithm for a special case where the form of an editing sequence is restricted. Indeed, insertions (or deletions) often occur at consecutive positions in practical cases. Thus we consider such a case in this section.

First we consider a simple case where editing operations must be applied in the following way (see Fig. 5), where $i_1 < i_2 < \cdots < i_k$ and $j_1 < j_2 < \cdots < j_k$ hold, and '$*$' means that an adequate character is used.

$$
\begin{array}{llll}
ins(1, j_1, *, \rightarrow), & ins(2, j_1, *, \rightarrow), & \cdots, & ins(i_1, j_1, *, \rightarrow), \\
ins(i_1+1, j_2, *, \rightarrow), & ins(i_1+2, j_2, *, \rightarrow), & \cdots, & ins(i_2, j_2, *, \rightarrow), \\
\cdots & & & \\
ins(i_{k-1}+1, j_k, *, \rightarrow), & ins(i_{k-1}+2, j_k, *, \rightarrow), & \cdots, & ins(i_k, j_k, *, \rightarrow), \\
ins(i_1, j_1+1, *, \uparrow), & ins(i_1, j_1+2, *, \uparrow), & \cdots, & ins(i_1, j_2, *, \uparrow), \\
ins(i_2, j_2+1, *, \uparrow), & ins(i_2, j_2+2, *, \uparrow), & \cdots, & ins(i_2, j_3, *, \uparrow), \\
\cdots & & & \\
ins(i_k, j_k+1, *, \uparrow), & ins(i_k, j_k+2, *, \uparrow), & \cdots, & ins(i_k, n, *, \uparrow), \\
sub(i_1', j_1', *), & sub(i_2', j_2', *), & \cdots, & sub(i_h', j_h', *).
\end{array}
$$

Note that the total cost of this editing sequence is $(i_k + (n - j_1))C_I + hC_S$. Then the problem is, given two-dimensional arrays $A[1..m, 1..n]$ and $B[1..m, 1..n]$, to find a minimum cost editing sequence S such that S has the above form (denoted by FORM-A) and S transforms $A[1..m, 1..n]$ to $B[1..m, 1..n]$.

This special problem can be solved in polynomial time by means of a reduction to the shortest path problem. First we construct a directed graph $G(V, E)$ as below:

$$
\begin{array}{rcl}
V & = & \{ (i,j) \mid 1 \le i \le m, 1 \le j \le n \} \cup \{START, GOAL\}, \\
E & = & \{ ((i,j),(i+1,j)) \mid 1 \le i < m, 1 \le j < n \} \cup \{ ((i,j),(i+1,j')) \mid 1 \le i < m, j < j' < n \} \cup \\
 & & \{ (START,(1,j)) \mid 1 \le j < n \} \cup \{ ((i,j), GOAL) \mid 1 < i \le m, 1 \le j < n \}.
\end{array}
$$

Next we define the cost of each edge. Let $LM(i,j)$ denote the number of mismatches between $A[i, 1..j]$ and $B[i, 1..j]$ where $A[i,k]$ corresponds to $B[i,k]$. Let $RM(i,j)$ denote the number of mismatches between $A[i+1, j..n-1]$ and $B[i, j+1..n]$ where $A[i+1,k]$ corresponds to $B[i,k+1]$. Then the cost of each edge is defined by:

$$cost(START, (1,j)) = 0,$$

$$cost((i,j), GOAL) = (n-j+1)C_I + (\, LM(i,j-1) + \sum_{k=i+1}^{m} LM(k,n)\,)C_S,$$

$$cost((i,j),(i+1,j)) = (LM(i,j-1) + RM(i,j))C_S + C_I,$$

$$cost((i,j),(i+1,j')) = (LM(i,j-1) + RM(i,j'))C_S + (j'-j+1)C_I \quad (j < j').$$

Then a shortest path from $START$ to $GOAL$ corresponds to a minimum cost editing sequence from $A[1..m, 1..n]$ to $B[1..m, 1..n]$. Therefore the minimum cost editing sequence can be found by solving the shortest path problem for a graph $G(V, E)$.

Here we consider the time complexity. The number of vertices of $G(V, E)$ is $O(mn)$ and the number of edges of $G(V, E)$ is $O(mn^2)$. $O(mn)$ time is sufficient for computing $LM(i,j)$'s and $RM(i,j)$'s in total. Thus the time complexity depends on the time for solving the shortest path problem. Since this graph has a special form, the shortest path problem can be solved in $O(mn^2)$ time using a dynamic programming technique as in the conventional string alignment algorithm. Note that $O(m^2n)$ time is sufficient even in the case of $m > n$ by slightly modifying the construction of a graph. Thus we have the following theorem.

Theorem 2: A minimum cost editing sequence can be found in $O(mn \min(m,n))$ time if each sequence must have FORM-A.

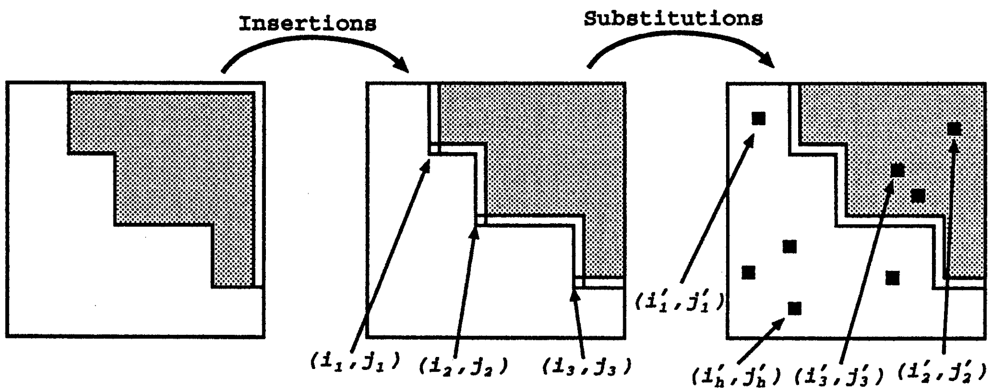

Figure 5: A simple case solved in polynomial time.

This algorithm can be extended for a more general case (see Fig. 6). We consider the case where each editing sequence must be a sequence of h sequences of insertions, each of which has the same form as FORM-A, followed by a sequence of substitutions. Moreover we assume that two sequences of insertions do not cross and h is bounded by some fixed constant k. It is a natural extension, and can be solved in a polynomial time in the following way.

In the algorithm for the original case, each vertex in $G(V, E)$ corresponds to a position (i, j) of an array. In the extended case, we construct a graph such that each vertex corresponds to a h-tuple $((i,j_1),(i,j_2),\cdots,(i,j_h))$ such that $h \le k$. Then the definitions of edges and costs are almost trivial, and omitted in this paper. In this case, the number of vertices and the number of edges are polynomially bounded since k is assumed to be a constant. Thus the size of $G(V, E)$ is polynomial of n and the shortest path problem for $G(V, E)$ can be solved in polynomial time. Therefore we can obtain a polynomial time algorithm in this extended case.

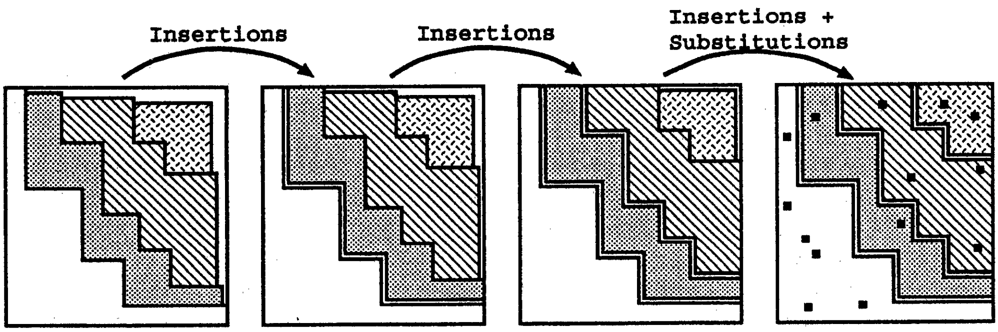

Figure 6: An extented case.

5 Conclusion

In this paper, we proposed an editing distance problem between two-dimensional arrays, in which rows are treated in the same way as in columns. We presented a polynomial time algorithm in a special case as well as a hardness result. However the presented results are preliminary ones. There remain many problems. For example, approximability, improvement of the algorithm, and other special and practical cases should be studied. Further studies may lead to practical two-dimensional pattern matching algorithms.

References

[1] A. Amir and M. Farach, "Efficient 2-dimensional approximate matching of non-rectangular figures," *Proc. ACM-SIAM Symp. Discrete Algorithms*, pp. 212–223, 1991.

[2] A. Amir and G. M. Landau, "Fast parallel and serial multidimensional approximate array matching," *Theoretical Computer Science*, vol. 81, pp. 97–115, 1991.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[4] K. Krithivasan and R. Sitalakshmi, "Efficient two-dimensional pattern matching in the presence of errors," *Information Sciences*, vol.43, pp. 169–184, 1987.

[5] G. M. Landau and U. Vishkin, "Fast parallel and serial approximate string matching," *J. Algorithms*, vol. 10, pp. 157-169, 1989.

[6] G. M. Landau and U. Vishkin, "Two dimensional pattern matching in a digitized image," *Proc. 4th Symp. Combinatorial Pattern Matching* (LNCS 684), pp. 134–151, 1993.

[7] K. Zhang, R. Statman and D. Shasha, "On the editing distance between unordered labeled trees," *Information Processing Letters*, vol. 42, pp. 133-139, 1992.

[8] K. Zhang, D. Shasha and J. T. L. Wang, "Approximate tree matching in the presence of variable length don't cares," *J. Algorithms*, vol. 16, pp. 33-66, 1994.