JOURNAL OF COMPLEXITY **15**, 128–147 (1999)
Article ID jcom.1998.0497, available online at http://www.idealibrary.com on IDEAL®

Efficient Algorithms for Approximate String Matching with Swaps*

Dong Kyue Kim

Department of Computer Engineering, Seoul National University, Seoul 151-742, Korea

Jee-Soo Lee

Department of Computer Science, Korea National Open University, Seoul 110-791, Korea

and

Kunsoo Park and Yookun Cho

Received June 26, 1997

Most research on the edit distance problem and the k-differences problem considered the set of edit operations consisting of changes, insertions, and deletions. In this paper we include the *swap* operation that interchanges two adjacent characters into the set of allowable edit operations, and we present an $O(t \min(m, n))$-time algorithm for the extended edit distance problem, where t is the edit distance between the given strings, and an $O(kn)$-time algorithm for the extended k-differences problem. That is, we add swaps into the set of edit operations without increasing the time complexities of previous algorithms that consider only changes, insertions, and deletions for the edit distance and k-differences problems. © 1999 Academic Press

1. INTRODUCTION

Given two strings $A[1 \cdots m]$ and $B[1 \cdots n]$ over an alphabet Σ, the *edit distance* between A and B is the minimum number of *edit operations* needed

* This research was supported by KOSEF Grant 981-0925-128-2.

to convert A into B. The edit distance problem is to find the edit distance between A and B. Most common edit operations are the following.

(i) *change*: replace one character of A by another single character of B.

(ii) *deletion*: delete one character from A.

(iii) *insertion*: insert one character into B.

These three edit operations are the ones commonly used in applications [EGGI92, SK83,WF74, WM92], though only insertions and deletions are considered in some work [My86]. A discrepancy between A and B that is corrected by an edit operation is called a *difference*.

The problem of string matching with *k-differences* (or the k-differences problem) is defined as follows: Given a pattern A of length m, a text B of length n, and an integer k, find all positions of B where A occurs with at most k differences.

Many algorithms have been developed for the edit distance problem and the k-differences problem [BN96, GG88]. When the edit distance t between A and B is small, an $O(t \min(m, n))$-time algorithm due to Ukkonen [Uk85] is the best one for the edit distance problem. When the given difference k is small, $O(kn)$-time algorithms due to Landau and Vishkin [LV89], Galil and Park [GP90], and Ukkonen and Wood [UW93] are best for the k-differences problem.

In this paper we consider an additional edit operation:

(iv) *swap*: interchange two adjacent characters in A.

The swap operation was first considered in [LW75, Wa75] and it is a special case of a *reversal* which is one of common genome rearrangements [HP95]. Lowrance and Wagner [LW75] proposed an $O(mn)$-time algorithm for the *extended* edit distance problem including the swap operation. For the k-differences problem the swap operation has never been considered. The k-differences problem including the swap operation will be called the *extended* k-differences problem.

Ukkonen [Uk85] considered *transpositions* in the edit distance problem. Galil and Park [GP90] also considered transpositions in the k-differences problem. A transposition is a correction of a difference that two adjacent characters in A correspond to two adjacent characters in B. However, the swap operation is more general than a transposition because deletions may occur before a swap and insertions may occur after a swap, but a transposition must not accompany deletions or insertions.

EXAMPLE 1. Let $A =$ `abcdeefg` and $B =$ `ahceegif`. The edit distance between A and B is four, as shown in Fig. 1a: The character `b` in A is

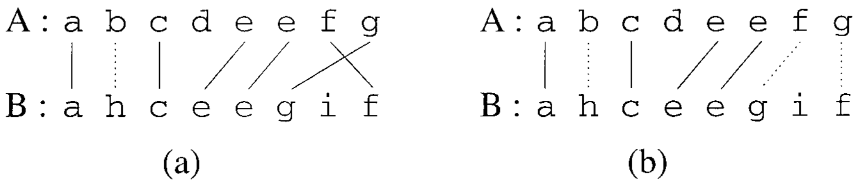

FIG. 1. The edit distances between *A* and *B*.

changed to h in *B*, d in *A* is deleted, f and g in *A* are swapped to g and f in *B*, and i in *B* is inserted. However, if only transpositions (but not swaps) are allowed, a transposition cannot be applied to fg in *A* because g and f in *B* are not adjacent. Hence the edit distance is five, as shown in Fig. 1b.

We present efficient algorithms for the extended edit distance problem and the extended k-differences problem. To compute the edit distance t between *A* and *B*, our algorithm takes $O(t \min(m, n))$ time, which is the same as that of Ukkonen [Uk85]. Our algorithm for the extended k-differences problem also takes $O(kn)$ time as in [GP90, LV89, UW93]. That is, we add swaps into the set of edit operations without increasing the time complexities of [GP90, LV89, Uk85, UW93] that consider only changes, insertions, and deletions for the edit distance and k-differences problems.

The paper is organized as follows. In the next section we describe the tables to compute the edit distance between two strings, i.e., the D-table, the C_D-table, and the H-table. In Section 3 we compute the C_H-table for the extended edit distance problem, and in Section 4 we present an efficient algorithm for the extended k-differences problem.

2. PRELIMINARIES

In this section we describe well-known algorithms for computing edit distances.

2.1. *D-Table and C_D-Table*

We will describe the D-table and the C_D-table for the edit distance problem between two strings *A* and *B* when the set of edit operations consists of change, deletion, and insertion.

Wagner and Fischer [WF74] devised an algorithm that takes $O(mn)$ time to compute the D-table. Let $D(i, j)$, $0 \leqslant i \leqslant m$ and $0 \leqslant j \leqslant n$, be the edit distance between $A[1 \cdots i]$ and $B[1 \cdots j]$. An entry $D(i, j)$ of the D-table is determined by the three entries $D(i-1, j-1)$, $D(i-1, j)$, and $D(i, j-1)$. The recurrence for the D-table is as follows: For all $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$,

$$D(i, j) = \min\{D(i-1, j-1) + \delta_{ij}, D(i-1, j) + 1, D(i, j-1) + 1\},$$

where $\delta_{ij} = 0$ if $A[i] = B[j]$; $\delta_{ij} = 1$ otherwise.

Ukkonen [Uk85] proved that $D(i, j) = D(i-1, j-1)$ or $D(i, j) = D(i-1, j-1) + 1$ in the D-table and proposed the C_D-table which provided a more compact way to store the information of the D-table.

Let D-diagonal d be the entries $D(i, j)$ such that $d = j - i$. For each D-diagonal we store only the positions where the values increase. For a D-diagonal d and a difference e, the entry $C_D(e, d)$ of the C_D-table is the largest column j such that $D(j-d, j) = e$. In other words, the entries of value e on D-diagonal d end at column $C_D(e, d)$. Note that $C_D(e, d) - d$ is the row of the last entry on D-diagonal d whose value is e.

The C_D-table is computed by the algorithm *Make-*C_D in Fig. 2, which is essentially Ukkonen's algorithm [GP90, Uk85]. We add special characters $\#_a$ and $\#_b$ (not in Σ) at the end of A and B, respectively, to simplify codes.

2.2. *H-Table*

In this subsection we describe the H-table for the extended edit distance problem [LW75].

Wagner [Wa75] showed that the extended edit distance problem is in general NP-complete and most of the restricted cases can be solved in polynomial time. Lowrance and Wagner [LW75] considered a restricted case of the problem, i.e., the cost of two swaps is at least as large as the sum of costs of an insertion and a deletion, and computed the edit distance in $O(mn)$ time by constructing the H-table. When we count the number of

```
Algorithm Make-$C_D$ ($A[1 \cdots m], B[1 \cdots n]$)
Initialize the $C_D$-table and set $A[m+1] = \#_a$, $B[n+1] = \#_b$, $e \leftarrow 0$
while $C_D(e, m-n) < n$ do
    for $d \leftarrow -e$ to $e$ do
        $c \leftarrow \max\{C_D(e-1, d-1)+1, C_D(e-1, d)+1, C_D(e-1, d+1)\}$
        $c' \leftarrow \min\{c, m+d, n\}$
        while $A[c'+1-d] = B[c'+1]$ do $c' \leftarrow c'+1$ od
        $C_D(e, d) \leftarrow c'$
    od
    $e \leftarrow e+1$
od
```

FIG. 2. Algorithm *Make-*C_D for constructing the C_D-table.

edit operations, the restriction above holds, and thus we can use the H-table. Let $H(i, j), 0 \leqslant i \leqslant m$ and $0 \leqslant j \leqslant n$, be the edit distance between $A[1 \cdots i]$ and $B[1 \cdots j]$ when swaps are added to the set of edit operations.

For $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$, let p_{ij} be the largest position less than i such that $A[p_{ij}] = B[j]$ and q_{ij} be the largest position less than j such that $A[i] = B[q_{ij}]$. Such positions p_{ij} and q_{ij} are called the *last-positions* for $H(i, j)$.

To compute an entry $H(i, j)$ of the H-table, we need the last-positions p_{ij} and q_{ij} [LW75]. When we apply a swap at $H(i, j)$, we should do the following (see Fig. 3):

(i) delete the characters from $p_{ij}+1$ to $i-1$ in A,

(ii) swap $A[p_{ij}](=B[j])$ and $A[i](=B[q_{ij}])$, and

(iii) insert the characters from $q_{ij}+1$ to $j-1$ in B.

Let *d-length* α_{ij} (resp. *i-length* β_{ij}) be the number of deleted (resp. inserted) characters to apply a swap at $H(i, j)$, i.e., $\alpha_{ij} = i - p_{ij} - 1$ and $\beta_{ij} = j - q_{ij} - 1$. Then the *swap-cost* $s(i, j)$ is

$$s(i, j) = H(p_{ij} - 1, q_{ij} - 1) + \alpha_{ij} + \beta_{ij} + 1.$$

The recurrence used in the H-table is

$$H(i, j) = \min\{H(i-1, j-1) + \delta_{ij}, H(i-1, j) + 1, H(i, j-1) + 1, s(i, j)\}. \tag{1}$$

If $s(i, j)$ is less than $\min\{H(i-1, j-1) + \delta_{ij}, H(i-1, j) + 1, H(i, j-1) + 1\}$, then we say that a swap *occurs at* $H(i, j)$.

3. THE EXTENDED EDIT DISTANCE PROBLEM

We will present a more efficient algorithm for the extended edit distance problem. To achieve $O(t \min(m, n))$ time complexity, our goal is to construct the C_H-table of two strings $A[1 \cdots m]$ and $B[1 \cdots n]$ when every edit operation has a unit cost. The C_H-table is to the H-table what the C_D-table is to the D-table. In this section we first show some properties of the H-table. Then we describe where to consider swaps and how to find the occurrences of swaps in the C_H-table.

3.1. *Properties of the H-Table*

We will define three types of swap operations and prove the diagonalwise monotonicity property in the H-table. When a swap occurs,

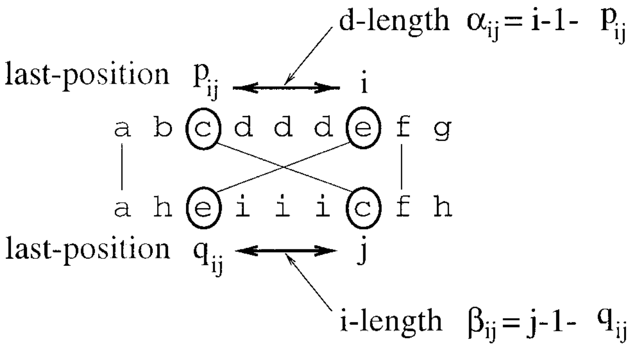

FIG. 3. The swap-cost $s(i, j)$ of a swap operation applying at $H(i, j)$.

deletions and/or insertions may occur between two swapped characters as Fig. 3 suggests. However, Lemma 1 shows that in case of unit costs a swap cannot accompany both deletions and insertions. Hence, we no longer need to consider those swaps that accompany both deletions and insertions. (Changes are at least as cheap as such swaps.)

LEMMA 1. *Let* α_{ij} *be the d-length and* β_{ij} *be the i-length for* $H(i, j)$. *If* $\alpha_{ij} > 0$ *and* $\beta_{ij} > 0$ *then* $s(i, j) \geqslant H(i-1, j-1) + 1$.

Proof. Let p_{ij} and q_{ij} be the last-positions for $H(i, j)$. By recurrence (1), $H(x-1, y-1)$, $H(x-1, y)$, and $H(x, y-1)$ are larger than or equal to $H(x, y) - 1$ for every $1 \leqslant x \leqslant m$ and $1 \leqslant y \leqslant n$. Hence, $H(p_{ij}-1, q_{ij}-1) \geqslant H(i-1, j-1) - \max\{i - p_{ij}, j - q_{ij}\}$. Since $\alpha_{ij} = i - p_{ij} - 1$ and $\beta_{ij} = j - q_{ij} - 1$, we have

$$\begin{aligned} s(i, j) &= H(p_{ij}-1, q_{ij}-1) + \alpha_{ij} + \beta_{ij} + 1 \\ &\geqslant H(i-1, j-1) - \max\{\alpha_{ij}+1, \beta_{ij}+1\} + \alpha_{ij} + \beta_{ij} + 1 \\ &= H(i-1, j-1) + \min\{\alpha_{ij}, \beta_{ij}\} \qquad (\alpha_{ij} > 0, \beta_{ij} > 0) \\ &\geqslant H(i-1, j-1) + 1. \quad \blacksquare \end{aligned}$$

By Lemma 1 there are three types of swaps occurring at $H(i, j)$:

(i) *transposition*: the case when $p_{ij} = i-1$ and $q_{ij} = j-1$. $A[i-1]$ and $A[i]$ are swapped to $B[j-1]$ and $B[j]$.

(ii) *d-swap*: the case when $p_{ij} < i-1$ and $q_{ij} = j-1$. After the characters $A[p_{ij}+1 \cdots i-1]$ are deleted, $A[p_{ij}]$ and $A[i]$ are swapped to $B[j-1]$ and $B[j]$.

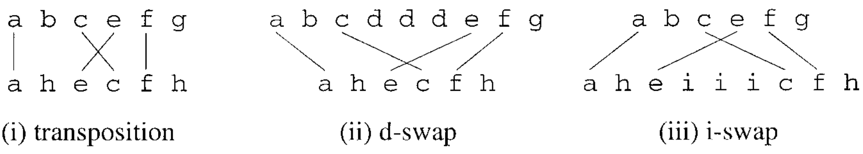

FIG. 4. Three types of swaps.

(iii) *i-swap*: the case when $p_{ij} = i-1$ and $q_{ij} < j-1$. $A[i-1]$ and $A[i]$ are swapped to $B[q_{ij}]$ and $B[j]$ and then the characters $B[q_{ij}+1 \cdots j-1]$ are inserted between them.

EXAMPLE 2. Figure 4 shows three types of swaps. The characters c and e in A are swapped to e and c in B.

Lemma 2 means diagonalwise monotonicity of the H-table, which implies that we can make the C_H-table for a H-table as we did the C_D-table for a D-table.

LEMMA 2. *$H(i, j) = H(i-1, j-1)$ or $H(i, j) = H(i-1, j-1)+1$ for every $i \geqslant 1$ and $j \geqslant 1$.*

Proof. We add the case of swaps into Ukkonen's proof [Uk85] for the D-table. Since $H(i, j)$ is always an integer, it suffices to show that $H(i, j) - 1 \leqslant H(i-1, j-1) \leqslant H(i, j)$. Recurrence (1) directly implies that $H(i, j)$ cannot be larger than $H(i-1, j-1)+1$. Hence, we have $H(i, j)-1 \leqslant H(i-1, j-1)$.

We will prove $H(i, j) \geqslant H(i-1, j-1)$ only for the case of swaps. Let α_{ij} be the d-length and β_{ij} be the i-length for $H(i, j)$. Recall that $s(i, j) \geqslant H(i-1, j-1) + \min\{\alpha_{ij}, \beta_{ij}\}$ in the proof of Lemma 1. Since a swap that goes with both insertions and deletions cannot occur by Lemma 1, $\min\{\alpha_{ij}, \beta_{ij}\} = 0$. Hence we have $s(i, j) \geqslant H(i-1, j-1)$, and therefore $H(i, j) \geqslant H(i-1, j-1)$ when a swap occurs at $H(i, j)$. ∎

COROLLARY 1. *If a swap occurs at $H(i, j)$, then $H(i, j) = H(i-1, j-1)$.*

Proof. If a swap occurs at $H(i, j)$, swap-cost $s(i, j)$ must be less than the cost of a change applied at $H(i, j)$. That is, $s(i, j) < H(i-1, j-1)+1$. By Lemma 2, we have $H(i, j) = s(i, j) = H(i-1, j-1)$. ∎

3.2. *Swap-Positions in the C_H-Table*

We will construct the C_H-table in the rest of Section 3. We first define the C_H-table and then describe where to consider swap operations during the computation of the C_H-table.

DEFINITION 1. Let H-diagonal d be the entries $H(i, j)$ such that $d = j - i$. For a H-diagonal d and a difference e, the entry $C_H(e, d)$ of the C_H-table is the largest column j such that $H(j - d, j) = e$.

We describe how to compute the C_H-table. Consider the computation of an entry $C_H(e, d)$. Assume by induction that $C_H(e-1, d-1)$, $C_H(e-1, d)$, and $C_H(e-1, d+1)$ were computed correctly. This means that in the H-table the entries of value $e-1$ reach column $C_H(e-1, d-1)$ on H-diagonal $d-1$, $C_H(e-1, d)$ on H-diagonal d, and $C_H(e-1, d+1)$ on H-diagonal $d+1$. Let c be the maximum of $C_H(e-1, d-1)+1$, $C_H(e-1, d)+1$, and $C_H(e-1, d+1)$. $H(c-d, c)$ gets value e from one of the last entries of value $e-1$ on H-diagonal $d-1$, d, and $d+1$ by one of insertion, change, and deletion, respectively. The entries of value e on H-diagonal d continue to the smallest column $c' \geqslant c$ such that $A[c' - d + 1] \neq B[c' + 1]$, and thus $C_H(e, d) = c'$.

Now we consider the swap operation. If $c' = c$ (i.e., $A[c - d + 1] \neq B[c + 1]$) then $H(c - d + 1, c + 1)$ would get value $e + 1$ without swaps. However, if a swap occurs at $H(c - d + 1, c + 1)$, the value of $H(c - d + 1, c + 1)$ can still be e. In this case, the entries of value e on H-diagonal d continue to the smallest column $c'' > c'$ such that $A[c'' - d + 1] \neq B[c'' + 1]$, and thus $C_H(e, d) = c''$. Hence, in the computation of $C_H(e, d)$ we must check whether any swaps occur at $H(c - d + 1, c + 1)$ or not.

DEFINITION 2. At the beginning of the computation of $C_H(e, d)$, let $c_i = C_H(e-1, d-1)+1$, $c_t = C_H(e-1, d)+1$, $c_d = C_H(e-1, d+1)$, and $c = \max\{c_i, c_t, c_d\}$. The positions $u = c - d + 1$ and $v = c + 1$ are called the *swap-positions* for $C_H(e, d)$.

There are three types of swaps occurring at $H(u, v)$:

(i) The case when $c = c_t$. Characters $A[u-1]$ and $B[v-1]$ have been changed to get $H(u-1, v-1)$. Hence we need to consider an occurrence of a *transposition* at $H(u, v)$.

(ii) The case when $c = c_d$. $A[u-1]$ has been deleted to get $H(u-1, v-1)$, and thus we need to consider an occurrence of a *d-swap* at $H(u, v)$.

(iii) The case when $c = c_i$. $B[v-1]$ has been inserted to get $H(u-1, v-1)$, and thus we need to consider an occurrence of an *i-swap* at $H(u, v)$.

To determine $C_H(e, d)$, we need to compute the swap-cost $s(u, v)$ for the three cases above. By Lemma 3, however, we need to compute $s(u, v)$ only when $c = c_t$. Proofs of Lemmas 3, 5–8 will be given in the appendix.

LEMMA 3. *Let u and v be the swap-positions for $C_H(e, d)$.*

(a) *If a d-swap occurs at* $H(u, v)$ *then* $c = c_d = c_t$.

(b) *If an i-swap occurs at* $H(u, v)$ *then* $c = c_i = c_t$.

3.3. D-*Swap and* i-*Swap Conditions*

We now describe how to find the occurrences of swaps in the C_H-table. We can easily find a transposition occurring at $H(u, v)$ for swap-positions $u = c - d + 1$ and $v = c + 1$. Since a transposition must come from H-diagonal d, it occurs at $H(u, v)$ if $A[u] = B[v-1]$, $A[u-1] = B[v]$, and $c = c_t$ [GP90, Uk85]. In this subsection we will describe the cases of d-swaps and i-swaps in the computation of $C_H(e, d)$.

To find whether or not a swap occurs, we need to compute the swap-cost $s(u, v)$, which in turn requires that we know last-positions p_{uv} and q_{uv}. Indeed, Lowrance and Wagner's algorithm [LW75] maintains all last-positions as it computes the H-table. However, a difficulty in the C_H-table is that all positions of A and B may not appear in the C_H-table. Instead of maintaining last-positions, we will find occurrences of swaps using two notions: "change-dominated" entries and "effectiveness" of insertions/deletions. Informally, an entry $H(i, j)$ is change-dominated if the value $H(i, j) = e$ is given by only a change operation in recurrence (1) (and not by any of a deletion, an insertion, a swap, and a match). A d-effectiveness (resp. i-effectiveness) indicates that a sequence of deletions (resp. insertions) takes place in A (resp. in B).

We now give formal definitions of the two notions. We first define two kinds of diagonals in the C_H-table. Let *A-diagonal* x be the entries $C_H(e, d)$ such that $e + d = x$ and *B-diagonal* y be the entries $C_H(e, d)$ such that $e - d = y$. See Fig. 5. Since the value of an entry in the C_H-table is a position of B, if $C_H(e, d) = j$ then we say that the position j of B *appears on B-diagonal* $e - d$. If $i = C_H(e, d) - d$ then we say that the position i of A *appears on A-diagonal* $e + d$.

Definition 3. An entry $H(i, j)$ of the H-table is *change-dominated* if

$$H(i, j) = H(i-1, j-1) + 1 < \min\{H(i-1, j) + 1, H(i, j-1) + 1, s(i, j)\}.$$

When $H(i, j)$ is change-dominated, we say that position i is *change-dominated on A-diagonal* $e + d$ and position j is *change-dominated on B-diagonal* $e - d$, where $e = H(i, j)$ and $d = j - i$.

Definition 4. Let u and v be swap-positions in the computation of $C_H(e, d)$.

(a) A position $p (\leqslant u - 2)$ in A is *d-effective* on A-diagonal $e + d$ if every position from p to $u - 2$ in A appears on A-diagonal $e + d$.

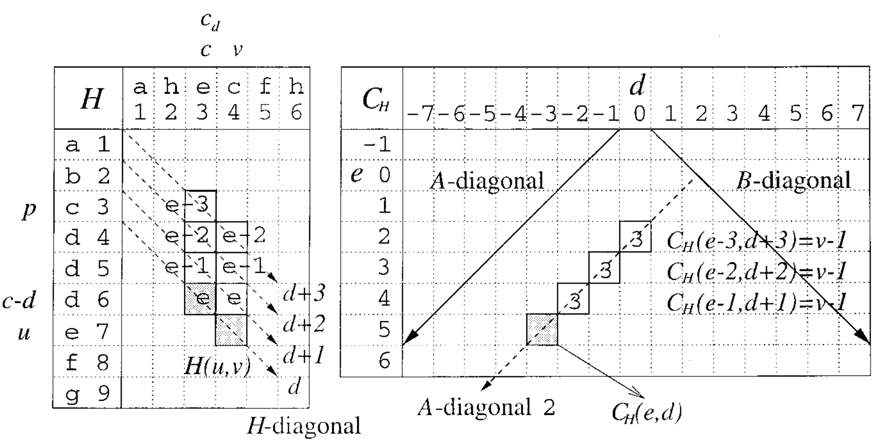

FIG. 5. The case when position 3 of A is d-effective on A-diagonal $e+d$ in the computation of $C_H(e,d) = C_H(5,-3)$ between two strings $A = $ abcdddefg and $B = $ ahecfh.

(b) A position $q(\leqslant v-2)$ in B is *i-effective* on B-diagonal $e-d$ if every position from q to $v-2$ in B appears on B-diagonal $e-d$.

Galil and Park [GP90] proposed a property of A-diagonals in Lemma 4. Based on Lemma 4, we will maintain some of the appeared positions on A-diagonals and B-diagonals.

LEMMA 4 [GP90]. *The positions of A (resp. B) that appear on the same A-diagonal (resp. B-diagonal) are strictly increasing in the C_H-table until the end position of A (resp. B).*

We will describe some properties related with d-effectiveness and i-effectiveness in Lemmas 5 and 6. Lemma 5 is used several times in proofs and by Lemma 6 we can check effectiveness in constant time.

LEMMA 5. (a) *A position p in A is d-effective on A-diagonal $e+d$ if and only if $H(u-1-i, v-1) = e-i$ and $H(u-i, v) = e-i+1$ for every $1 \leqslant i \leqslant u-1-p$.*

(b) *A position q in B is i-effective on B-diagonal $e-d$ if and only if $H(u-1, v-1-j) = e-j$ and $H(u, v-j) = e-j+1$ for every $1 \leqslant j \leqslant v-1-q$.*

LEMMA 6. (a) *A position p in A is d-effective on A-diagonal $e+d$ if and only if $C_H(e-a, d+a) = c$ where $a = u-1-p$.*

(b) *A position q in B is i-effective on B-diagonal $e-d$ if and only if $C_H(e-b, d-b) = q$ where $b = v-1-q$.*

Lemma 7 states a necessary and sufficient condition for occurrences of swaps. Let p_{uv} and q_{uv} be the last-positions for $H(u, v)$. Lemma 7 shows that a d-swap or an i-swap occurs when p_{uv} or q_{uv} is effective and $H(p_{uv}, q_{uv})$ is change-dominated. (See Fig. 6.)

LEMMA 7. *Suppose that $c = c_t$ and $A[u] \neq B[v]$.*

(a) *A d-swap occurs at $H(u, v)$ if and only if p_{uv} is d-effective on A-diagonal $e + d$, $q_{uv} = v - 1$, and $H(p_{uv}, q_{uv})$ is change-dominated.*

(b) *An i-swap occurs at $H(u, v)$ if and only if $p_{uv} = u - 1$, q_{uv} is i-effective on B-diagonal $e - d$, and $H(p_{uv}, q_{uv})$ is change-dominated.*

By Lemma 7, we can check whether a d-swap or an i-swap occurs or not if we maintain last-positions. However, in our algorithm we maintain change-dominated positions instead of last-positions, by which we can save space. Lemma 8 shows another condition that can replace the condition of Lemma 7. Moreover, this new condition can be checked in constant time using the following arrays.

We use two arrays LA and LB of size $(2t + 1)$ each, where t is the edit distance between A and B. Array LA will be used for applying d-swaps and array LB for i-swaps. In the computation of entry $C_H(e, d)$, we define arrays LA and LB as follows. An element $LA[x]$ is p if and only if p is the largest position less than $c_d - d$ in A such that p is change-dominated and appears on A-diagonal $x = e + d$. An element $LB[y]$ is q if and only if q is the largest position less than c_i in B such that q is change-dominated and appears on B-diagonal $y = e - d$. (The largest position that appears on A-diagonal $e + d$ (resp. B-diagonal $e - d$) is $c_d - d - 1$ (resp. $c_i - 1$). (See (F3) in the appendix.)

LEMMA 8. *Let $p = LA[e + d]$ and $q = LB[e - d]$.*

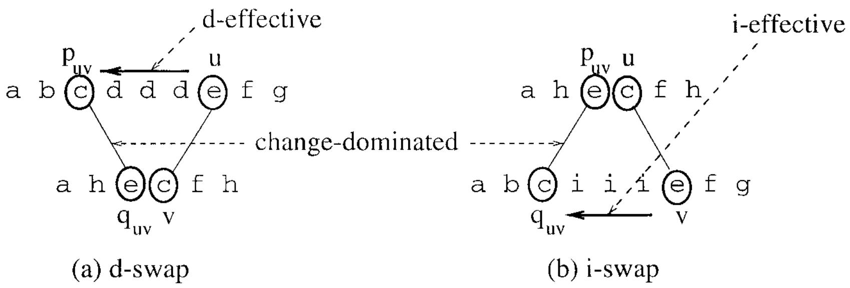

FIG. 6. The case when a d-swap or an i-swap occurs at $H(u, v)$.

(a) *$A[p] = B[v]$, $A[u] = B[v-1]$, and p is d-effective on A-diagonal $e+d$ if and only if p_{uv} is d-effective on A-diagonal $e+d$, $q_{uv} = v-1$, and $H(p_{uv}, q_{uv})$ is change-dominated.*

(b) *$A[u] = B[q]$, $A[u-1] = B[v]$, and q is i-effective on B-diagonal $e-d$ if and only if $p_{uv} = u-1$, q_{uv} is i-effective on B-diagonal $e-d$, and $H(p_{uv}, q_{uv})$ is change-dominated.*

THEOREM 1. *Suppose that $c = c_t$ and $A[u] \neq B[v]$. Let $a = u-1-p$ and $b = v-1-q$ when $p = LA[e+d]$ and $q = LB[e-d]$.*

(a) *A d-swap occurs at $H(u,v)$ if and only if $A[p] = B[v]$, $C_H(e-a, d+a) = c$, and $A[u] = B[v-1]$.*

(b) *An i-swap occurs at $H(u,v)$ if and only if $A[u] = B[q]$, $C_H(e-b, d-b) = q$, and $A[u-1] = B[v]$.*

Proof. It follows immediately from Lemmas 6, 7, and 8. ▮

3.4. *An Extended Edit Distance Algorithm*

We present algorithm *Make-C_H* in Fig. 7 for constructing the C_H-table. Algorithm *Make-C_H* works as algorithm *Make-C_D* does except the parts for swap operations. We initialize the C_H-table and arrays LA and LB. For each H-diagonal d and a difference e, we compute c. If $c = c_t$ and $A[u] \neq B[v]$, then we determine whether a swap occurs or not by Theorem 1. We set $a = \min\{e, u-1-p\}$ and $b = \min\{e, v-1-q\}$ because $u-1-p$ or $v-1-q$ may be larger than e, in which case no swaps occur at $H(u,v)$ because swap-cost $s(u,v) > e$. If one of a transposition (at line 11), a d-swap (at line 12), and an i-swap (at line 13) occurs, we increase c. Then we follow the procedure of *Make-C_D* and compute $C_H(e,d)$.

To perform $C_H(e,d) \leftarrow c'$ at line 18, it must be guaranteed that $H(c'+1-d, c'+1) = e+1$. Since other cases are simple, we only show that if there is at least one match at line 17 (i.e., $c' > c$ and $A[c'-d] = B[c']$) then no swaps occur at $H(c'+1-d, c'+1)$. Let $u' = c'+1-d$ and $v' = c'+1$. Since $A[u'-1] = B[v'-1]$, a transposition cannot occur at $H(u', v')$. In order for a d-swap or an i-swap to occur, a sequence of deletions or insertions must take place before the swap as shown in Fig. 6. We claim that $H(u'-2, v'-1) > e-1$ and $H(u'-1, v'-2) > e-1$. Otherwise, c should have been at least $c'(=v'-1)$ at lines 5 and 6, which is a contradiction because $c' > c$. Thus, $H(u'-1, v'-1)$ cannot get value e by a deletion or an insertion. Therefore, a d-swap or an i-swap cannot occur at $H(u', v')$.

After the computation of $C_H(e,d) = c'$, we update arrays LA and LB. If $c = c_t$ and $c > \max\{c_d, c_i\}$ then $H(c-d, c)$ is change-dominated by Definitions 1, 2, and 3 and Corollary 1. If $c' = c$ then positions $c-d$ and c appear on A-diagonal $e+d$ and B-diagonal $e-d$, respectively. Hence, in case of

```
**Algorithm** $Make\text{-}C_H$ $(A[1 \cdots m+1], B[1 \cdots n+1])$
1 Initialize the $C_H$-table.
2 Clear arrays $LA$ and $LB$, and set $e \leftarrow 0$
3 while $C_H(e, m-n) < n$ do
4       for $d \leftarrow -e$ to $e$ do
5           $c_t \leftarrow C_H(e-1, d)+1$, $c_d \leftarrow C_H(e-1, d+1)$, $c_i \leftarrow C_H(e-1, d-1)+1$
6           $c \leftarrow \max\{c_t, c_d, c_i\}$
7           $u \leftarrow c-d+1$, $v \leftarrow c+1$ // set swap-positions
8           if $c = c_t$ and $A[u] \neq B[v]$ then // consider swap operations
9              $p \leftarrow LA[e+d]$, $q \leftarrow LB[e-d]$
10             $a \leftarrow \min\{e, u-1-p\}$, $b \leftarrow \min\{e, v-1-q\}$
11             if ( $A[u] = B[v-1]$ and $A[u-1] = B[v]$ ) or
12                ( $A[p] = B[v]$ and $C_H(e-a, d+a) = c$ and $A[u] = B[v-1]$ ) or
13                ( $A[u] = B[q]$ and $C_H(e-b, d-b) = q$ and $A[u-1] = B[v]$ )
14                then $c \leftarrow c+1$ fi
15          fi
16          $c' \leftarrow \min\{c, m+d, n\}$
17          while $A[c'+1-d] = B[c'+1]$ do $c' \leftarrow c'+1$ od
18          $C_H(e, d) \leftarrow c'$
19          if $c' = c_t$ and $c' > \max\{c_d, c_i\}$ then // update arrays $LA$ and $LB$
20             $LA[e+d] \leftarrow c'-d$
21             $LB[e-d] \leftarrow c'$
22          fi
23      od
24      $e \leftarrow e+1$
25 od
```

FIG. 7. Algorithm *Make*-C_H for constructing the C_H-table.

$c' = c_t$ and $c' > \max\{c_d, c_i\}$, we set $LA[e+d] = c'-d$ and $LB[e-d] = c'$ (at lines 20 and 21).

EXAMPLE 3. Let $A =$ abcdddefg and $B =$ ahecfh. Figure 8 shows the H-table and the C_H-table of A and B. The extended edit distance between A and B is six. Consider the computation of $C_H(5, -3)$. We have $c_t = C_H(4, -3)+1 = 3$, $c_d = C_H(4, -2) = 3$, and $c_i = C_H(4, -4)+1 = 2$. Since $c = \max\{c_t, c_d, c_i\} = 3$, $u = c-d+1 = 7$ and $v = c+1 = 4$.

• *An occurrence of a d-swap.* $C_H(5, -3)$ is on A-diagonal $e+d = 2$. Consider $C_H(2, 0)$. Because $C_H(1, 0)+1 = 3$ and $\max\{C_H(1, -1)+1 = 2, C_H(1, 1) = 2\} < 3$, position 3 $(= C_H(2, 0) - 0)$ of A is change-dominated on A-diagonal 2. Hence $LA[2]$ has position $p = 3$. Let $a = u-1-p = 3$. Since $C_H(e-a, d+a) = C_H(2, 0) = 3$, $A[3] = B[4] =$ c, and $A[7] = B[3] =$ e, we have found a d-swap occurring at $H(7, 4)$ by Theorem 1.

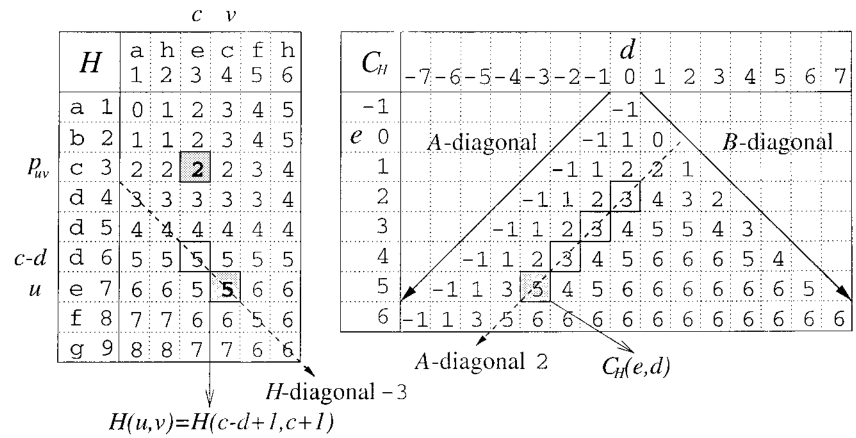

FIG. 8. The H-table and the C_H-table of strings `abcdddefg` and `ahecfh`.

Since swap-cost $s(7,4) = H(2,2) + (7-3-1) + 1 = 5$, we have $H(7,4) = s(7,4) = 5$.

- *The computation of* $C_H(5, -3)$. Since we found a d-swap, we increase c. Then $c' = 4$. Because there is a continuing match at column 5 on H-diagonal -3 (i.e., $A[8] = B[5] = \mathtt{f}$), we increase c'. Since there is a mismatch at column 6 (i.e., $A[9] \neq B[6]$), the value of $C_H(5, -3)$ becomes 5.

- *Updating LA and LB.* After we compute $C_H(5, -3) = c' = 5$, we check the conditions to update LA and LB. Since $c' > c_t = c_d = 3$ in this case, we need not update elements $LA[2]$ and $LB[8]$.

THEOREM 2. *Algorithm Make-C_H solves the extended edit distance problem in $O(t \min(m,n))$ time, where t is the edit distance between A and B.*

Proof. Algorithm *Make-C_D* takes $O(t \min(m,n))$ time and $O(t^2)$ space. To check whether or not a swap occurs, it takes constant time to perform lines 7–15 and lines 19–22 for each entry in the C_H-table. Hence algorithm *Make-C_H* also takes $O(t \min(m,n))$ time. ▮

4. THE EXTENDED k-DIFFERENCES PROBLEM

In this section we present an algorithm for the *extended k-differences* problem. Landau and Vishkin [LV89], Galil and Park [GP90], and Ukkonen and Wood [UW93] proposed $O(kn)$-time algorithms for the k-differences problem when the set of edit operations consists of changes, deletions, and insertions. Here we do not mention preprocessing because preprocessings are all different in [GP90, LV89, UW93] and preprocessing time is absorbed into $O(kn)$ when n is sufficiently larger than m.

We can apply the method used in Section 3 to any of the three algorithms. The algorithms can proceed A-diagonal by A-diagonal since the maximal difference k is given. There are $(n-m+1+k)$ A-diagonals in the C_H-table, but we need one variable LA only for all A-diagonals because the C_H-table is computed A-diagonal by A-diagonal.

We maintain array LB of size $(2k+1)$. When we compute an entry $C_H(e, x-e)$ on A-diagonal x, we need the element $LB[y]$ such that $y = 2e-x$. Since $0 \leqslant e \leqslant k$, we should keep the positions of B associated with the entries on B-diagonal y such that $-x \leqslant y \leqslant 2k-x$. Since $(2k+1)$ B-diagonals are needed for each A-diagonal, let $b = y \bmod (2k+1)$. Then we can use $LB[b]$ as the array LB in the algorithm for the extended k-differences problem. After the computation of $C_H(k, x-k)$ on each A-diagonal x, we do not need $LB[2k-x]$ any more and need $LB[-(x+1)]$. Let $b' = (2k-x) \bmod (2k+1) = (-x-1) \bmod (2k+1)$. Hence we reset variable LA and element $LB[b']$ in order to use them for the next A-diagonal $x+1$ and B-diagonal $-x-1$.

Example 4. Figure 9 shows the C_H-table when $k=3$ and there are 18 A-diagonals. The dotted parallelogram is the region where the entries of the C_H-table exist. Consider the computation of the entries on A-diagonal $x=10$. Since $-10 = -x \leqslant y \leqslant 2k-x = -4$, we maintain array LB for B-diagonals in the range of $[-10, -4]$. After the computation of $C_H(3, 7)$, we reset LA and $LB[3]$.

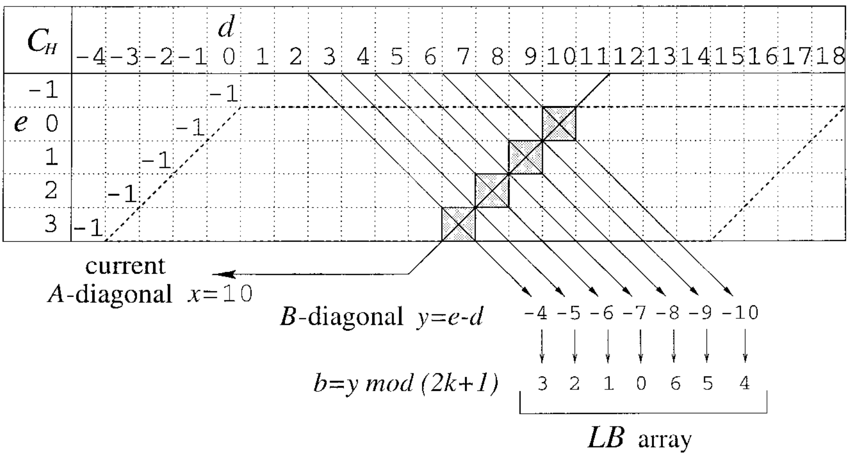

FIG. 9. The C_H-table and array LB for the extended k-differences problem.

THEOREM 3. *The extended k-differences problem can be solved in $O(kn)$ time, not including preprocessing.*

5. CONCLUSION

We have presented efficient algorithms for the extended edit distance and k-differences problems. By applying dynamic programming techniques with the C_H-table, we added swaps into the set of edit operations without increasing the time complexities of previous algorithms that consider only changes, insertions, and deletions for the edit distance and k-differences problems. It will be interesting to consider swaps in various applications of approximate string matching.

APPENDIX: PROOFS OF LEMMAS

First we list some facts during the computation of $C_H(e, d)$ that will be used in the following proofs.

(F1) $H(c-d, c) = e$.

(F2) $u = c-d+1$ and $v = c+1$.

(F3) The largest position of A (resp. B) that appears on A-diagonal $e+d$ (resp. B-diagonal $e-d$) is $C_H(e-1, d+1)-(d+1) = c_d-(d+1)$ (resp. $C_H(e-1, d-1) = c_i - 1$).

(F4) $\alpha_{uv} = u - p_{uv} - 1$ for d-length α_{uv} and last-position p_{uv}.

Facts (F1) and (F2) directly come from Definition 2. Fact (F3) follows from Lemma 4 because the largest appearing position in A (resp. B) corresponds to the previous entry on A-diagonal $e+d$ (resp. B-diagonal $e-d$). Fact (F4) is the relation between a last-position and its d-length.

Proof of Lemma 3. In the following proofs, we will prove (a) only because (b) is analogous. If a d-swap occurs at $H(u, v)$, then $c = c_d$. Hence $c = c_d \geqslant c_t$.

Suppose that $c_d > c_t$. Since $c_d = C_H(e-1, d+1)$ and $c_t = C_H(e-1, d)+1$, we have

$$H(c_d - (d+1), c_d) = e-1 \tag{2}$$

and $H(c_t - d, c_t) = e$ by Definition 1. Consider $H(c_t-(d+1), c_t)$. By recurrence (1), $H(c_t-(d+1), c_t) \geqslant H(c_t-d, c_t)-1 = e-1$. Since $c_d > c_t$, $H(c_t-(d+1), c_t) \leqslant H(c_d-(d+1), c_d) = e-1$ by Lemma 2. Hence

$$H(c_t-(d+1), c_t) = e-1. \tag{3}$$

From (2) and (3), the value of every entry from $H(c_t-(d+1), c_t)$ to $H(c_d-(d+1), c_d)$ on H-diagonal $d+1$ must be $e-1$. Note that $H(u-2, v-1)=H(c_d-(d+1), c_d)$ by (F2) and $c_d=c$. Since $c_d>c_t$, the previous entry $H(u-3, v-2)$ on H-diagonal $d+1$ has value $e-1$, too. The swap-cost $s(u, v)$ of the d-swap occurring at $H(u, v)$ is

$$\begin{aligned} s(u, v) &= H(p_{uv}-1, v-2)+\alpha_{uv}+1 \\ &\geqslant \{H(u-3, v-2)-(u-3-p_{uv}+1)\}+(u-1-p_{uv})+1 \\ &= H(u-3, v-2)+2 = H(u-2, v-1)+2 \geqslant H(u-1, v-1)+1. \end{aligned}$$

Because swap-cost $s(u, v)$ is at least as large as the cost of a change occurring at $H(u, v)$, a d-swap cannot occur at $H(u, v)$ and this is a contradiction. Therefore, we have $c_d=c_t$ when a d-swap occurs at $H(u, v)$. ▮

Proof of Lemma 5. (If) Since $H(u-1-i, v-1)=e-i$ and $H(u-i, v)=e-i+1$, we have $C_H(e-i, d+i)=v-1$ for every $1\leqslant i\leqslant u-1-p$ by Definition 1. See Fig. 5. Hence every position from p to $u-2$ in A appears on A-diagonal $e+d$, and thus p is d-effective on A-diagonal $e+d$.

(Only if) If p is d-effective, positions $p, ..., u-2$ in A appear on A-diagonal $e+d$. To show $H(u-1-i, v-1)=e-i$ and $H(u-i, v)=e-i+1$ for every i, we will prove $C_H(e-i, d+i)=v-1$. We proceed by induction on i. In order for $u-2(=c-d-1)$ to appear on A-diagonal $e+d$, c_d must be equal to c because the largest position that appears on A-diagonal $e+d$ is $c_d-(d+1)$ by (F3) and $c_d-(d+1)\leqslant c-d-1$ by $c_d\leqslant c$. Hence $C_H(e-1, d+1)=c_d=c=v-1$. Assume that $C_H(e-i, d+i)=v-1$ for $i\geqslant 1$. Let $c'=C_H(e-i', d+i')$ where $i'=i+1$. Before the computation of $C_H(e-i, d+i)$, the largest position that appears on A-diagonal $e+d$ is $c'-d-i'$. In order for $u-1-i'(=c-d-i')$ to appear on A-diagonal $e+d$, we must have $c'=c$ because $c'-(d+i')\leqslant c-d-i'$ by $c'\leqslant c$. Hence $C_H(e-i', d+i')=c=v-1$. ▮

Proof of Lemma 6. (Only if) If p is d-effective then $H(u-1-a, v-1)=e-a$ and $H(u-a, v)=e-a+1$ on H-diagonal $v-(u-a)=d+a$ by Lemma 5. Hence $C_H(e-a, d+a)=v-1=c$ by Definition 1.

(If) We show that $H(u-1-i, v-1)=e-i$ and $H(u-i, v)=e-i+1$ for every $1\leqslant i\leqslant a$ if $C_H(e-a, d+a)=v-1=c$. The position of A that appears on A-diagonal $e+d$ corresponding to $C_H(e-a, d+a)$ is $v-1-(d+a)=u-1-a=p$. Then we have $H(p, v-1)=e-a$ by Definition 1. As there are $a+1$ entries from $H(p, v-1)$ to $H(u-1, v-1)$ on column $v-1$ in the H-table and $H(x-1, y)\geqslant H(x, y)-1$ for every x and y by recurrence (1), we have $H(u-1-i, v-1)=e-i$ for every i.

To show that $H(u-i, v)=e-i+1$, suppose that there exist some $1<l<a$ such that $H(u-l, v)=e-l$. Then, $H(u-j, v)\leqslant e-j$ for every $1\leqslant j\leqslant l$ by recurrence (1). Since $H(u-1-j, v-1)=e-j$, $H(u-j, v)=e-j$ by Lemma 2. By Definition 1, $C_H(e-1, d+1)\geqslant v(=c+1)$, which is a contradiction because $c\geqslant c_d=C_H(e-1, d+1)$. Hence $H(u-i, v)=e-i+1$ for every i by Lemma 2. Therefore, p is d-effective on A-diagonal $e+d$ by Lemma 5. ▌

Proof of Lemma 7. (Only if) When a d-swap occurs at $H(u, v)$, characters $A[p_{uv}]$ and $A[u]$ will be swapped to adjacent characters $B[v-1]$ and $B[v]$. Thus, $A[u]=B[v-1]$ and we have $q_{uv}=v-1$ by definition of last-positions. To show that p_{uv} is d-effective, we will prove $C_H(e-\alpha_{uv}, d+\alpha_{uv})=v-1$ by Lemma 6 and (F4). That is, we will show $H(p_{uv}, v-1)=e-\alpha_{uv}$ and $H(p_{uv}+1, v)=e-\alpha_{uv}+1$.

When a d-swap occurs at $H(u, v)$, characters from $p_{uv}+1(=u-\alpha_{uv})$ to $u-1$ in A should be deleted. That is,

$$H(u-1, v-1)=H(u-2, v-1)+1=\cdots=H(u-1-\alpha_{uv}, v-1)+\alpha_{uv}.$$

Since $H(u-1, v-1)=e$ by (F1) and (F2),

$$H(p_{uv}, v-1)=H(u-1-\alpha_{uv}, v-1)=e-\alpha_{uv}.$$

Suppose that $H(p_{uv}+1, v)=e-\alpha_{uv}$. Then as in the proof of Lemma 6, we would have $C_H(e-1, d+1)\geqslant v$, which is a contradiction. Hence we have $H(p_{uv}+1, v)=e-\alpha_{uv}+1$ by Lemma 2.

We will prove that $H(p_{uv}, q_{uv})$ is change-dominated. Since $q_{uv}=v-1$, we want to show that

$$\begin{aligned}H(p_{uv}, v-1)&=H(p_{uv}-1, v-2)+1\\&<\min\{H(p_{uv}, v-2)+1, H(p_{uv}-1, v-1)+1, s(p_{uv}, v-1)\}\end{aligned}$$

by Definition 3. Recall that $H(p_{uv}, v-1)=e-\alpha_{uv}$ and $H(p_{uv}+1, v)=e-\alpha_{uv}+1$ as shown above.

Since p_{uv} is d-effective, $H(p_{uv}+1, v-1)=e-\alpha_{uv}+1$ by Lemma 5. By recurrence (1), we have $H(p_{uv}+1, v-1)\leqslant H(p_{uv}, v-2)+1$. Then,

$$H(p_{uv}, v-1)=e-\alpha_{uv}<H(p_{uv}+1, v-1)\leqslant H(p_{uv}, v-2)+1.$$

Since $A[p_{uv}]=B[v]$, we have $H(p_{uv}, v)=H(p_{uv}-1, v-1)$. Thus

$$\begin{aligned}H(p_{uv}, v-1)=e-\alpha_{uv}<H(p_{uv}+1, v)&\leqslant H(p_{uv}, v)+1\\&=H(p_{uv}-1, v-1)+1.\end{aligned}$$

Since $H(p_{uv}, v-1) < H(p_{uv}, v-2)+1$ and $H(p_{uv}, v-1) < H(p_{uv}-1, v-1)+1$, $H(p_{uv}, v-1)$ cannot come from an insertion or a deletion. Any swaps must not occur at $H(p_{uv}, v-1)$, because a character can be swapped at most once [LW75]. Moreover, as $A[p_{uv}] = B[v] \neq A[u] = B[v-1]$, $H(p_{uv}, v-1)$ cannot come from a match. Hence, $H(p_{uv}, v-1)$ can get the minimum value $e-\alpha_{uv}$ only by a change. Therefore, $H(p_{uv}, v-1) = H(p_{uv}-1, v-2)+1$ and $H(p_{uv}, q_{uv})$ is change-dominated.

(If) We show that a d-swap occurs at $H(u, v)$ if p_{uv} is d-effective on A-diagonal $e+d$, $q_{uv} = v-1$ and $H(p_{uv}, q_{uv})$ is change-dominated. Recall that $H(u-1, v-1) = e$. Since $A[u] \neq B[v]$, $\delta_{uv} = 1$. Then $H(u, v)$ cannot have value e by a match and would get value $e+1$ by a change.

Since p_{uv} is d-effective on A-diagonal $e+d$, $H(u-1, v) = e$ by Lemma 5. Hence $H(u, v)$ would get value $e+1$ by a deletion.

Since $c_i \leqslant c$, $C_H(e-1, d-1) = c_i - 1 < c-1 = v-2$. By $A[u] = B[v-1]$ (i.e., $q_{uv} = v-1$), we have $H(u-1, v-2) = H(u, v-1)$ on H-diagonal $d-1$. This implies that $C_H(e-1, d-1) < v-2$ by Definition 1, and thus $H(u-1, v-2) \geqslant e$. Since $H(u, v-1) \geqslant e$ by Lemma 2, $H(u, v)$ would be $H(u, v-1)+1 \geqslant e+1$ by an insertion.

However, consider the swap-cost $s(u, v)$. Since p_{uv} is d-effective, we have $H(p_{uv}, v-1) = e-\alpha_{uv}$ by Lemma 5. Since $H(p_{uv}, q_{uv})$ is change-dominated,

$$s(u, v) = H(p_{uv}-1, q_{uv}-1) + \alpha_{uv} + 1 = H(p_{uv}, v-1) + \alpha_{uv} = e.$$

Hence $s(u, v) < \min\{H(u-1, v-1) + \delta_{uv}, H(u-1, v)+1, H(u, v-1)+1\}$, and therefore a d-swap occurs at $H(u, v)$. ∎

Proof of Lemma 8. $A[u] = B[v-1]$ if and only if $q_{uv} = v-1$ by definition of last-positions. Now we will show that $A[p] = B[v]$ and p is d-effective if and only if p_{uv} is d-effective and $H(p_{uv}, v-1)$ is change-dominated.

(Only if) Since p is d-effective, $H(u-1-i, v-1) = e-i$ and $H(u-i, v) = e-i+1$ for every $1 \leqslant i \leqslant a = u-1-p$ by Lemma 5. This implies that no match can occur at $H(u-i, v)$ for every i, i.e., $A[u-i] \neq B[v]$. Because $A[p] = B[v]$, p is the largest position less than u such that $A[p] = B[v]$, and hence p is the last-position p_{uv}. Therefore, p_{uv} is d-effective and also change-dominated on A-diagonal $e+d$ by definition of array LA. Because $H(p_{uv}, v-1) = e-\alpha_{uv}$ by Lemma 5 and $v-1-p_{uv} = v-1-(u-1-\alpha_{uv}) = d+\alpha_{uv}$, $H(p_{uv}, v-1)$ is change-dominated by Definition 3.

(If) Since p_{uv} is d-effective, $H(u-j-1, v-1) = e-j$ for every $1 \leqslant j \leqslant \alpha_{uv} = u-1-p_{uv}$ by Lemma 5. That is, $H(u-j-1, v-1) = H(u-j-2, v-1)+1$ for $1 \leqslant j < \alpha_{uv}$, and thus $H(u-\alpha_{uv}, v-1), ..., H(u-2, v-1)$ cannot be change-dominated by Definition 3. (Note that $v-1-(u-j-1) = d+j$.) It means that positions $p_{uv}+1 (= u-\alpha_{uv}), ..., u-2$ cannot be change-dominated on A-diagonal $e+d$. Because $H(p_{uv}, v-1)$ is change-dominated, p_{uv} is the largest position less than $u-1(=c-d=c_d-d)$ such that p_{uv} is change-dominated and appears on A-diagonal $e+d$, and thus $p_{uv}=p$ by definition of array LA. Therefore, p is d-effective and $A[p]=B[v]$ by definition of last-positions. ▮

ACKNOWLEDGMENT

We thank the referees for their helpful comments.

REFERENCES

[BN96] Baeza-Yates, R., and Navarro, G. (1996), A faster algorithm for approximate string matching, *in* "Proc. of the 7th Symp. on Combinatorial Pattern Matching," Lectures Notes in Comput. Sci., Vol. 1075, pp. 1–23, Springer-Verlag, New York/Berlin.

[EGGI92] Eppstein, D., Galil, Z., Giancarlo, R., and Italiano, G. (1992), Sparse dynamic programming. I. Linear cost functions, *J. Assoc. Comput. Mach.* **39**, 519–545.

[GG88] Galil, Z., and Giancarlo, R. (1998), Data structures and algorithms for approximate string matching, *J. Complexity* **4**, 33–72.

[GP90] Galil, Z., and Park, K. (1990), An improved algorithm for approximate string matching, *SIAM J. Comput.* **19**, 989–999.

[HP95] Hannenhalli, S., and Pevzner, P. A. (1995), Transforming men into mice (polynomial algorithm for genomic distance problem), *in* "IEEE Sympos. Found. Computer Science," pp. 581–592.

[LV89] Landau, G. M., and Vishkin, U. (1989), Fast parallel and serial approximate string matching, *J. Algorithms* **10**, 157–169.

[LW75] Lowrance, R., and Wagner, R. A. (1975), An extension of the string-to-string correction problem, *J. Assoc. Comput. Mach.* **22**, 177–183.

[My86] Myers, E. W. (1986), An *O*(*ND*) difference algorithm and its variations, *Algorithmica* **1**, 251–266.

[SK83] Sankoff, D., and Kruskal, J. B. (1983), "Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison," Addison–Wesley, New York.

[Uk85] Ukkonen, E. (1985), Algorithms for approximate string matching, *Inform. and Control* **64**, 100–118.

[UW93] Ukkonen, E., and Wood, D. (1993), Approximate string matching with suffix automata, *Algorithmica* **10**, 353–364.

[Wa75] Wagner, R. A. (1975), On the complexity of the extended string-to-string correction problem, *in* "ACM Symp. Theory of Computing," pp. 218–223.

[WF74] Wagner, R. A., and Fischer, M. J. (1974), The string-to-string correction problem, *J. Assoc. Comput. Mach.* **21**, 168–173.

[WM92] Wu, S., and Manber, U. (1992), Fast text searching allowing errors, *Comm. Assoc. Comput. Mach.* **35**, 83–91.