Journal of Complexity 26 (2010) 364–374

Contents lists available at ScienceDirect

Journal of Complexity

journal homepage: www.elsevier.com/locate/jco

Hardness of comparing two run-length encoded strings☆

Kuan-Yu Chen [a], Ping-Hui Hsu [a], Kun-Mao Chao [a,b,c,*]

[a] Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan
[b] Graduate Institute of Biomedical Electronics and Bioinformatics, National Taiwan University, Taipei 106, Taiwan
[c] Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei 106, Taiwan

ARTICLE INFO

Article history:
Received 3 June 2009
Accepted 24 March 2010
Available online 7 April 2010

Keywords:
Compressed pattern matching
Run-length encoding
Sequence comparison

ABSTRACT

In this paper, we consider a commonly used compression scheme called run-length encoding. We provide both lower and upper bounds for the problems of comparing two run-length encoded strings. Specifically, we prove the 3SUM-hardness for both the wildcard matching problem and the k-mismatch problem with run-length compressed inputs. Given two run-length encoded strings of m and n runs, such a result implies that it is very unlikely to devise an $o(mn)$-time algorithm for either of them. We then present an inplace algorithm running in $O(mn \log m)$ time for their combined problem, i.e. k-mismatch with wildcards. We further demonstrate that if the aim is to report the positions of all the occurrences, there exists a stronger barrier of $\Omega(mn \log m)$-time, matching the running time of our algorithm. Moreover, our algorithm can be easily generalized to a two-dimensional setting without impairing the time and space complexity.

1. Introduction

The explosion of digital data urges the need for devising effective data compression techniques. Run-length encoding (abbreviated as RLE) is one of the best-known coding schemes that performs lossless data compression. RLE compression simply represents the consecutive, identical symbols of a string with a run, usually denoted by σ^i, where σ is an alphabet symbol and i is its repetition

☆ A preliminary version of this work appeared in the 20th Annual Symposium on Combinatorial Pattern Matching, France, 2009.

* Corresponding author at: Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan.
E-mail address: kmchao@csie.ntu.edu.tw (K.-M. Chao).

0885-064X/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jco.2010.03.003

times. For example, string *bbcccddaaaaa* can be compressed into RLE format as $b^2c^3d^2a^5$. Because of its simplicity and efficiency, run-length encoding is widely used in several areas. In fax transmission, RLE compression is combined with other techniques into Modified Huffman Coding [15]. Since faxed documents are typically black texts on a white background, RLE compression is particularly suitable for them and often achieves good compression ratios. For a similar reason, this coding is also largely applied in optical character recognition, in which the inputs are usually images of large scales of identically valued pixels [20]. Other applications appear in bioinformatics, where RLE compression is employed to speed up the comparison of two biological sequences [10,14,18].

In 1992, Amir and Benson [2] initiated the study of the so-called *compressed pattern matching problem*, in which the aim is to access (search) the compressed file without decompressing it. In particular, they studied the two-dimensional pattern matching problem with the text compressed into RLE format. Since then this has been an active research field, and several papers have delved into compressed pattern matching problems under various compression schemes (e.g., RLE compression, LZ-family compression, or straight-line programs). For the RLE scheme, some papers took one step further by considering problems of comparing two RLE strings using different cost functions, such as the LCS metric [5,17,19], the Levenshtein distance [16,18], and arbitrary alignment scores [10,14]. In this paper, we investigate the hardness of comparing (or approximately matching) two strings both compressed into the RLE format.

This paper is organized as follows. In Section 2, we compare our results with the related work and introduce the base problems used in later reductions. In Section 3, we prove the 3SUM-hardness for the *wildcard matching problem* and the *k-mismatch problem* with run-length compressed inputs. In Section 4.1, we provide an upper bound for their combined problem, i.e. the *k-mismatch with wildcards problem*, and further demonstrate, in Section 4.2, that, if the aim is to report the positions of all the occurrences, the problem is at least as hard as sorting pairwise sums. Section 4.3 generalizes our algorithm to a two-dimensional setting. Section 5 concludes the paper by mentioning some open problems.

2. Preliminaries

2.1. Notation

Throughout the paper, we adopt the following notation. We let P and T denote the original string representation of the pattern and the text, which are compressed into the RLE strings P_c and T_c, respectively. All the problems we investigate in the paper will take P_c and T_c as the input. We let capital letters M and N denote the lengths of P and T, and let small letters m and n denote the number of runs of P_c and T_c. We let $P[i]$ (resp., $T[i]$) denote the i-th symbol of P (resp., of T). Furthermore, given two strings A and B, AB denotes the string obtained by appending B to the end of A. Similarly, given two RLE strings A_c and B_c, A_cB_c denotes the RLE string obtained by appending B_c to the end of A_c.

2.2. Related work and our results

For the wildcard matching problem and the k-mismatch problem, previous work all focused on the uncompressed string inputs. (The formal definitions of these two problems can be found in Sections 3.1 and 3.2, respectively.) The fastest existing algorithms for them both rely on the technique of Fast Fourier Transforms (FFTs). The fastest algorithm for the wildcard matching problem runs in $O(N \log M)$ time [8], while the fastest algorithm for the k-mismatch problem runs in $O(N\sqrt{k \log k})$ time [4]. As for their combined problem, i.e. the k-mismatch with wildcards problem, there exists an $O(N\sqrt{M \log M})$-time solution by extending the algorithm proposed in [1]. Very recently, Clifford et al. [9] gave an $O(Nk \log^2 M(\log^2 k + \log \log M))$-time algorithm, which is more efficient for small k. For these problems, we will show that if the input strings are compressed into the RLE format of m and n runs, it is very unlikely to solve them in $o(mn)$ time. We then provide an upper bound of $O(mn \log m)$ time and $O(m)$ extra space.

The advantage of our algorithm is two-fold. On the one hand, it is a "fully" compressed matching algorithm, meaning that it can cope with inputs where both the pattern and the text are compressed.

This allows us to directly manipulate the compressed data, now lying in disks in the RLE format, without any effort to decompress them. On the other hand, given two uncompressed strings, since one can always compress them into the RLE format in $O(M+N)$ time and then run our algorithm, our result implies an $O(M+N+mn\log m)$-time solution for uncompressed strings. For cases where the compression ratio is good enough, i.e. when $O(mn\log m)$ is $O(M+N)$, our algorithm matches the trivial lower bound of $O(M+N)$-time, and thus outperforms all the existing algorithms mentioned above. Meanwhile, our solution is an *inplace* algorithm (a notion proposed in [3]), meaning that it uses little extra space in proportional to the compressed pattern size, and can be generalized to solve the *approximate* counterpart of the two-dimensional matching problem considered in [2,3].

It should be noted that the reductions we establish in the paper apply to the problems in a simpler setting where the alphabet is finite, whereas the algorithm we present is capable of handling input strings over an infinite alphabet. Moreover, for simplicity's sake, we do not assume *optimal encoding* on the input RLE strings. That is, RLE strings such as $b^2c^1c^2d^2a^3a^2$ are legitimate inputs to the problems. All our results, however, can be easily applied to problems that take as input the RLE strings with optimal encoding.

2.3. The 3SUM problem

The 3SUM problem is to decide whether there exists a triple a, b, c in a set of n integers such that $a+b+c=0$. Devising a $\Theta(n^2)$-time algorithm for 3SUM is easy, but it turns out to be difficult to further improve this time bound. Thus, the 3SUM problem serves as a base problem for a class of problems conjectured to require $\Omega(n^2)$ time [12]. In this paper, we will use a variant of the 3SUM problem, denoted by 3SUM', as the base problem of our reductions. The 3SUM and 3SUM' problems are linearly reducible to each other [12].

Problem 1. 3SUM': Given three sets of integers A, B, and C, each of size n, are there $a\in A$, $b\in B$, and $c\in C$ with $a+b=c$?

We say problem PR is 3SUM-hard if an instance of 3SUM can be reduced to an instance of PR in $o(n^2)$ time. We follow the notation of [12]. Given problems PR1 and PR2, we write PR1 $\ll_{f(n)}$ PR2 if every instance of PR1 of size n can be solved by answering an instance of PR2 of size $O(n)$ with additional $f(n)$ time. Thus, problem PR is 3SUM-hard if 3SUM $\ll_{o(n^2)}$ PR. Many problems, especially in computational geometry, have been shown to fall in the class of 3SUM-hard problems [6,12]. Below, we introduce a known 3SUM-hard problem called *discrete segment-containing-points* (abbreviated DISCRETE-SCP). Note that this problem is a discrete version of the SCP problem considered in [6], where the authors cope with real values. Following the paradigm of [6], one can easily show that DISCRETE-SCP is also 3SUM-hard. Both 3SUM' and DISCRETE-SCP will serve as base problems in our later reductions.

Problem 2. DISCRETE-SCP: Given a set U of m integers and a set V of n pairwise-disjoint intervals, where $m=O(n)$, is there an integer number (translation) u such that $U+u\subseteq V$? Here, an interval is a set of consecutive integers $\{i, i+1, \ldots, j\}$, denoted by $[i, j]$.

In principle, when proving lower bounds by reductions, we tend to define the target problems as simply as possible without diminishing their 3SUM-hardness (in a sense that the base problem is approached by simplifying the target problems). In doing so, we not only establish a closer relationship between the problems, but also the target problems are easier to serve as base problems in the reductions to other problems. Therefore, in Section 3 we first consider *decision problems* with a *finite alphabet*. In contrast, when presenting the upper bound, we try to define the problem as generally as possible (in order to demonstrate the strength of the algorithm). Thus, in Section 4 we show how our algorithm can be applied to a *combined problem* with an *infinite alphabet*.

3. Lower bounds

There exists an optimal linear-time solution, a by-product of [11], for the exact string matching problem with RLE inputs. In this section, we demonstrate that if one aims at finding an "approximate" occurrence, the problem is at least as hard as the 3SUM problem. There are two commonly used

"approximate" criteria in the literature: (1) introducing a special symbol called a *wildcard* into the alphabet and (2) allowing several mismatches between the occurrence and the pattern. The former is known as the wildcard matching problem, and the latter is known as the k-mismatch problem.

3.1. The wildcard matching problem

The wildcard symbol, usually denoted by $*$, is a symbol that can match any symbol in the alphabet Σ. Let pattern P and text T be two strings over $\Sigma \cup \{*\}$. Pattern P is said to occur at position i in T if, for every position j in the pattern, either $P[j] = T[i + j - 1]$ or at least one of $P[j]$ and $T[i + j - 1]$ is a wildcard. We formally define the wildcard matching problem with RLE inputs, denoted by RLE-WILDCARD, as follows.

Problem 3. RLE-WILDCARD: Assume that pattern P and text T, two strings over $\Sigma \cup \{*\}$, are compressed into the RLE strings P_c and T_c, respectively. The problem is, given P_c and T_c, to locate all occurrences of P in T.

The construction of the 3SUM-hardness for RLE-WILDCARD is effortless if we establish the reduction from DISCRETE-SCP. The idea is to encode the instances of DISCRETE-SCP as strings, and use symbols 1's to represent points and symbols 0's or wildcards to represent the gap between points (a similar idea appears in [7]).

Theorem 1. DISCRETE-SCP $\ll_{n \log n}$ RLE-WILDCARD.

Proof. Given an instance U and V of DISCRETE-SCP of sizes m and n, we construct two RLE strings P_c and T_c over alphabet $\Sigma = \{0, 1\}$ as follows. We first sort U and V in $O(n \log n)$ time. Let $U' = \langle p_1, p_2, \ldots, p_m \rangle$, $p_i < p_{i+1}$, be the sorted sequence of U, and $V' = \langle [q_1, r_1], [q_2, r_2], \ldots, [q_n, r_n] \rangle$, $q_i \le r_i < q_{i+1} \le r_{i+1}$, be the sorted intervals of V. We next construct two RLE strings, P_c and T_c, as an instance of RLE-WILDCARD:

$$P_c = 1^1 *^{p_2-p_1-1} 1^1 *^{p_3-p_2-1} \ldots 1^1 *^{p_m-p_{m-1}-1} 1^1,$$
$$T_c = 1^{r_1-q_1+1} 0^{q_2-r_1-1} 1^{r_2-q_2+1} 0^{q_3-r_2-1} \ldots 0^{q_n-r_{n-1}-1} 1^{r_n-q_n+1}.$$

Note that P_c and T_c contain $2m - 1$ and $2n - 1$ runs, respectively. Let P and T be the uncompressed string representations of P_c and T_c. It is easily seen that there is a translation u such that $U + u \subseteq V$ if and only if there is an occurrence of P in T. □

The proof of Theorem 1 shows that the problem is 3SUM-hard even if the wildcards are restricted to appear in the pattern. What remains interesting is whether the problem with wildcards only appearing in the text, which we denote by RLE-WILDCARD$_T$, is also 3SUM-hard. For completeness' sake, we establish the 3SUM-hardness for this case as follows. The proof needs more elaboration and is an adaptation of the idea used in [6].

Theorem 2. 3SUM' $\ll_{n \log n}$ RLE-WILDCARD$_T$.

Proof. Given (A, B, C), an instance of 3SUM', without loss of generality we assume that they contain only positive numbers. By sorting A, B, and C we obtain their sorted sequences $A' = \langle a_1, a_2, \ldots, a_n \rangle$, $B' = \langle b_1, b_2, \ldots, b_n \rangle$, and $C' = \langle c_1, c_2, \ldots, c_n \rangle$. Let $d = \max\{x \mid x \in A \cup B \cup C\} + 1$. Again, we construct two RLE strings, P_c and T_c, as an instance of RLE-WILDCARD$_T$ (see Fig. 1 for an illustration):

$$P_c = 0^{d-a_n} 1^1 0^{a_n-a_{n-1}-1} 1^1 \ldots 0^{a_2-a_1-1} 1^1 0^{a_1+b_1+d-2} 1^1 0^{b_2-b_1-1} 1^1 \ldots 0^{b_n-b_{n-1}-1} 1^1 0^{d-b_n},$$
$$T_c = *^{d-1} 1^1 *^{d+c_1-2} 1^1 *^{d-1} 1^1 *^{d+c_2-2} 1^1 \ldots *^{d+c_n-2} 1^1 *^{d-1}.$$

We have that both P_c and T_c contain $2n + 1$ runs. Let P and T be the uncompressed string representations of P_c and T_c. Below, we show that there is a triple $a \in A, b \in B, c \in C$ with $a + b = c$ if and only if there is an occurrence of P in T.

Let α_i (resp., β_j) denote the symbol 1, in P, that corresponds to number a_i (resp., b_j). Let γ_k and γ'_k denote the first and the second 1's, in T, that correspond to number c_k. The only-if direction is simple. If there exists a triple $a_i + b_j = c_k$, we can find an occurrence of P in T by aligning α_i with γ_k and β_j with γ'_k. As for the other direction, observe that P is of length $3d$ and any substring of length $3d$ in T

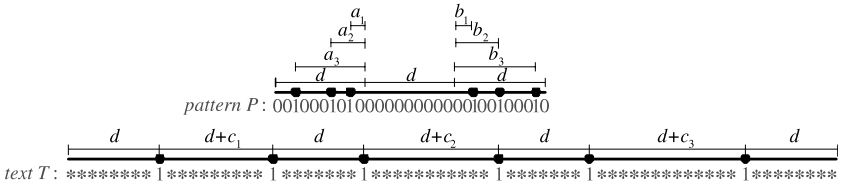

Fig. 1. The construction of pattern P and text T for the 3SUM reduction of the wildcard matching problem with wildcards only appearing in the text. This figure depicts an example with sorted sets $A = \{a_1, a_2, a_3\}$, $B = \{b_1, b_2, b_3\}$, and $C = \{c_1, c_2, c_3\}$.

must contain at least two 1's. The occurrence cannot contain γ'_{k-1} and γ_k for some k, since they are separated by $d-1$ symbols and no pair of 1's in P is of the same gap length. This leads us to the other case where the occurrence contains γ_k and γ'_k for some k. Therefore, we have that γ_k and γ'_k must be aligned with some α_i and β_j in P, implying that $a_i + b_j = c_k$. □

3.2. The k-mismatch problem

Given two strings $A = a_1 a_2 \ldots a_\ell$ and $B = b_1 b_2 \ldots b_\ell$ of the same length, the Hamming distance of A and B is defined as $d_H(A, B) = \ell - |S|$, where $S = \{(i, i) \mid a_i = b_i, 1 \le i \le \ell\}$. For simplicity, we let $d_H(A, B) = \infty$ if $|A| \neq |B|$. Now we define the k-mismatch problem with RLE inputs, denoted by RLE-MISMATCH, as follows.

Problem 4. RLE-MISMATCH: Assume that pattern P and text T, two strings over Σ, are compressed into the RLE strings P_c and T_c, respectively. The problem is, given P_c, T_c, and an arbitrary number K, to locate all substrings T' of T such that $d_H(P, T') \le K$.

Let P and T be an instance of RLE-WILDCARD, and let P' be the string obtained by replacing every wildcard symbol in P with an additional symbol \$. By setting parameter K to the total number of \$'s in P', the answer to whether P' occurs in T with at most K mismatches can be used to determine whether P occurs in T. Hence, the 3SUM-hardness of the decision problem of RLE-MISMATCH follows immediately from Theorem 1.

Corollary 1. RLE-WILDCARD $\ll_{m+n}$ RLE-MISMATCH.

Note that the established lower bound on RLE-MISMATCH is for an arbitrary K. That is, we do not prove, for example, that the problem is 3SUM-hard when $K = 1$. Moreover, the range for parameter K is between 0 and $|P|$, and thus an $o(Kn)$-time solution for RLE-MISMATCH may exist and is not ruled out by Corollary 1.

4. An upper bound of the k-mismatch with wildcards problem

So far we have revealed the $\Omega(mn)$-time barrier in solving RLE-WILDCARD and RLE-MISMATCH. In this section, we give an $O(mn \log m)$-time algorithm for their combined problem. Assume that pattern P and text T are strings over $\Sigma \cup \{*\}$, where Σ is possibly infinite, and both of them are compressed into the RLE strings $P_c = X_1 X_2 \ldots X_m$ and $T_c = Y_1 Y_2 \ldots Y_n$, where X_i and Y_j are the i-th and j-th runs of P_c and T_c, respectively. Given P_c, T_c, and an arbitrary number K, we now show how to find all substrings T' of T such that $d_H(P, T') \le K$.[1]

4.1. A plane-sweep approach

Definition 1. We define D to be an $M \times N$ matrix whose entry $D[i, j] = \delta(P[i], T[j])$, where $\delta(a, b) = 0$ if symbol a matches symbol b and $\delta(a, b) = 1$ otherwise. For those entries $D[i, j]$ where $j - i = d$, they are said to be on diagonal d of D.

[1] It should be noted that our algorithm in fact solves a more general problem whose goal is to compute the Hamming distance of the pattern at all text positions.

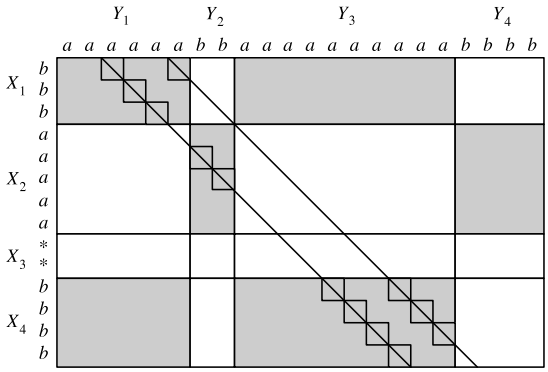

Fig. 2. The matrix D defined according to pattern $P_c = b^3a^5*^2b^4$ and text $T_c = a^6b^2a^{10}b^4$. Observe that diagonals 2 and 5 contain nine and four entries that are 1's, which implies that there are nine and four mismatches if the pattern is slid to the text positions 3 and 6, respectively. Moreover, matrix D is partitioned into mismatch blocks (grey blocks) and match blocks (white blocks).

Definition 2. For each diagonal d, we define $F(d)$ to be the number of entries that are 1's on diagonal d of D. That is, $F(d) = \sum_{i=1}^{M} D[i, i+d]$ for $d \in [-M+1, N-1]$.

Observe that to see where P occurs in T it suffices to identify those diagonals $d \in [0, N-M]$ with $F(d) \leq K$ (see Fig. 2 for an illustration). To compute the values of F, instead of accumulating the number of 1's diagonal-by-diagonal, we do it "block-by-block". Note that each run pair X_i and Y_j corresponds to a sub-matrix of D, which we denote by $D_{i,j}$. Since all entries in $D_{i,j}$ are 0's if X_i and Y_j encode the same symbol or either one encodes a wildcard, and are 1's otherwise, we can partition matrix D into mismatch blocks (of 1's) and match blocks (of 0's). For each mismatch block $D_{i,j}$, we define function $f_{i,j}$ as follows.

Definition 3. For each mismatch block $D_{i,j}$, we define $f_{i,j}(d)$ to be the number of entries of sub-matrix $D_{i,j}$ on diagonal d of D, where $d \in [-M+1, N-1]$.

Observe that $F(d) = \sum f_{i,j}(d)$ for $d \in [-M+1, N-1]$. Taking Fig. 2 for example, we have $f_{1,1}(2) = 3, f_{2,2}(2) = 2, f_{4,3}(2) = 4$, and $F(2) = f_{1,1}(2) + f_{2,2}(2) + f_{4,3}(2) = 9$. Let $(x_{i,j}, y_{i,j})$ and $(x'_{i,j}, y'_{i,j})$ denote the upper-left and lower-right corners of mismatch block $D_{i,j}$. If $y_{i,j} - x_{i,j} \leq y'_{i,j} - x'_{i,j}$, the values of $f_{i,j}(d)$ can be easily calculated by the following formula (see Fig. 3 for the diagram of $f_{i,j}$).

$$f_{i,j}(d) = \begin{cases} 0, & \text{for } -M-1 \leq d < y_{i,j} - x'_{i,j} - 1; \\ d - y_{i,j} + x'_{i,j} + 1, & \text{for } y_{i,j} - x'_{i,j} - 1 \leq d < y_{i,j} - x_{i,j}; \\ x'_{i,j} - x_{i,j} + 1, & \text{for } y_{i,j} - x_{i,j} \leq d < y'_{i,j} - x'_{i,j}; \\ y'_{i,j} - x_{i,j} - d + 1, & \text{for } y'_{i,j} - x'_{i,j} \leq d < y'_{i,j} - x_{i,j} + 1; \\ 0, & \text{for } y'_{i,j} - x_{i,j} + 1 \leq d \leq N+1. \end{cases}$$

The values of $f_{i,j}$ for the other case where $y_{i,j} - x_{i,j} > y'_{i,j} - x'_{i,j}$ can be similarly calculated. To simplify the discussions, we assume there exist dummy diagonals $-M-1$, $-M$, N, and $N+1$, and thus the domain of F and $f_{i,j}$ is extended from $[-M+1, N-1]$ to $[-M-1, N+1]$. We further let $f_{i,j}(-M-1) = f_{i,j}(-M) = f_{i,j}(N) = f_{i,j}(N+1) = 0$ and $F(-M-1) = F(-M) = F(N) = F(N+1) = 0$. We define the "turning points" of a function as follows.

Definition 4. Given a function $f : [-M-1, N+1] \to \mathbb{N}$, we define the forward difference operator of f to be function $\Delta f(d) = f(d+1) - f(d)$ for $d \in [-M-1, N]$. We say that position $d \in [-M, N]$ is a "turning point" of f if $\Delta f(d) \neq \Delta f(d-1)$.

Adding the dummy diagonals ensures the presence of the first and the fourth turning points of all $f_{i,j}$. We view $y_{i,j} - x_{i,j}$ and $y'_{i,j} - x'_{i,j}$ as two distinct turning points even when they have an identical value. Hence, we have that each $f_{i,j}$ contains exactly four distinct turning points. Recall that these

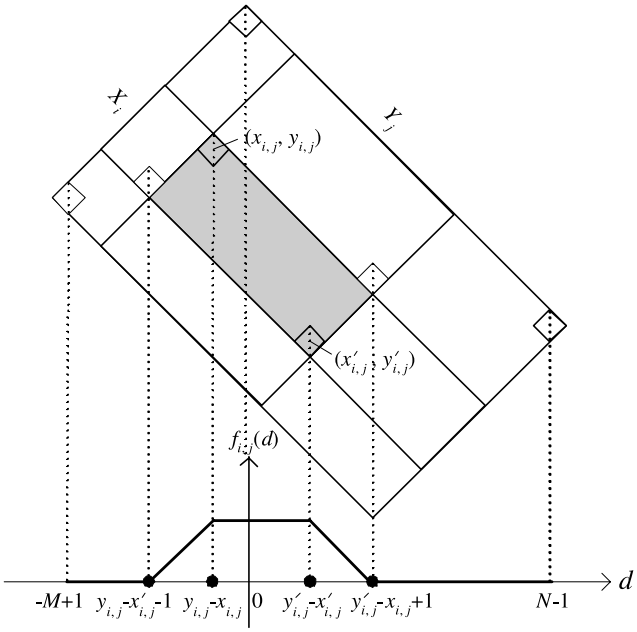

Fig. 3. The diagram of $f_{i,j}$ for the case where $y_{i,j} - x_{i,j} \le y'_{i,j} - x'_{i,j}$. The values of $f_{i,j}$ are contained in five connected line segments of slopes 0, 1, 0, -1, 0 from left to right, respectively. Note that the horizontal line segments may be absent if $y_{i,j} - x'_{i,j} - 1 \le -M + 1$ or $y_{i,j} - x_{i,j} = y'_{i,j} - x'_{i,j}$ or $y'_{i,j} - x_{i,j} + 1 \ge N - 1$.

Algorithm FINDCHANGE
Input: Two RLE strings $P_c = X_1X_2 \dots X_m$ and $T_c = Y_1Y_2 \dots Y_n$.
Output: A set S of ordered pairs (d, δ), where d denotes the position of a turning point of some $f_{i,j}$ and δ denotes the increment (or decrement) of $\Delta f_{i,j}$ at position d.
Initialization: $S \leftarrow \phi$;
1 **for each** mismatch block $D_{i,j}$ **do**
2 Compute the upper-left corner $(x_{i,j}, y_{i,j})$ and the lower-right corner $(x'_{i,j}, y'_{i,j})$ of $D_{i,j}$;
4 $S \leftarrow S \cup (y_{i,j} - x'_{i,j} - 1, 1) \cup (y_{i,j} - x_{i,j}, -1) \cup (y'_{i,j} - x'_{i,j}, -1) \cup (y'_{i,j} - x_{i,j} + 1, 1)$;
5 **end for**
6 Output S;

Fig. 4. The algorithm for computing all the turning points generated by the mismatch blocks.

turning points indicate the positions at which the value of $\Delta f_{i,j}$ changes, and the increments (or decrements) of $\Delta f_{i,j}$ at these positions are $+1$, -1, -1, and $+1$ from left to right, respectively. Algorithm FINDCHANGE (Fig. 4) computes the turning points of all $f_{i,j}$ and their corresponding increments (or decrements) of $\Delta f_{i,j}$.

Now we present the algorithm, ACCUMULATING (Fig. 5), for computing the values of function F. It is a plane-sweep approach based on the following observation: $\Delta F(d) = F(d+1) - F(d) = \sum f_{i,j}(d+1) - \sum f_{i,j}(d) = \sum \Delta f_{i,j}(d)$ for $d \in [-M-1, N]$. Hence, if we scan the values of ΔF from left to right, we have that initially $\Delta F(-M-1) = \sum \Delta f_{i,j}(-M-1) = 0$ and the value of ΔF changes whenever some $\Delta f_{i,j}$ changes. This means that a turning point of some $f_{i,j}$ is also a turning point of function F. The algorithm therefore simulates a vertical line scanning from left to right, and stops (performs actions) whenever it meets a turning point. More precisely, it first sorts all the turning points according to their x-coordinates (line 1). Throughout the procedure, it keeps track of variables *pos* and *slope*, which denote the current position of the scan-line and the current value of ΔF, respectively. At each iteration of the while-loop (lines 2–8), the algorithm retrieves the next turning point, computes the value of F at that position, and then updates the values of *pos* and

Algorithm ACCUMULATING
Input: Set S from algorithm FINDCHANGE.
Output: A list L of the intersections of the diagram of F from left to right.
Initialization: $pos \leftarrow -M-1$; $slope \leftarrow y \leftarrow 0$;
1 Sort S according to the first coordinate. Let S' be the sorted list;
2 **while** S' is not exhausted **do**;
3 Retrieve the next pair (d, δ) from list S';
4 **if** $d \neq pos$ **then**
5 $y \leftarrow y + (d - pos) \times slope$;
7 Insert (d, y) into the end of list L;
6 $pos \leftarrow d$;
8 **end if**
9 $slope \leftarrow slope + \delta$;
10 **end while**
11 Output L;

Fig. 5. A plane-sweep approach for accumulating the number of entries that are 1's on each diagonal of D.

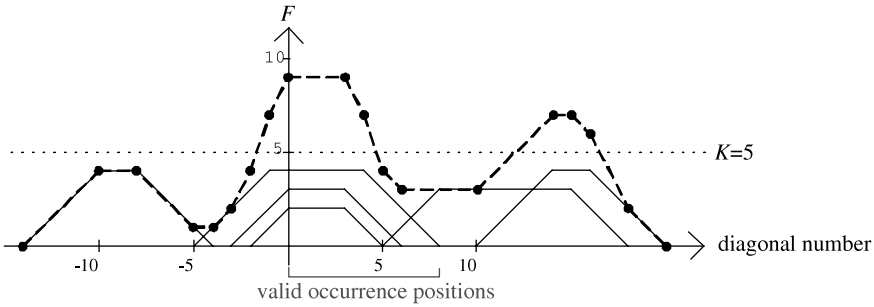

Fig. 6. The accumulated diagram of F (the dashed lines). Our algorithm computes the intersections of the diagram of F from left to right. The solid lines depict the diagrams of $f_{i,j}$ of all mismatch blocks $D_{i,j}$, and the dotted line specifies a threshold, $K = 5$, for the tolerated mismatches. This figure continues the example of Fig. 2, in which the pattern occurs at the text positions 6, 7, 8, and 9.

slope accordingly. As a result, the algorithm computes the values of F at all its turning points (see Fig. 6 for an example). Once the intersections of the diagram of F are computed, it is easy to output the positions of all the occurrences. Clearly, the algorithm spends $O(mn \log mn)$ time (because of the sorting procedure) and requires $O(mn)$ extra space (due to the size of list S).

When devising a compressed matching algorithm, the amount of extra space used is often another restricting factor [3]. We say that a compressed matching algorithm is "inplace" if the extra space used in the algorithm is proportional to the input pattern size, which is $O(m)$ in our problems. Below, we show that our algorithm can be further refined into an *inplace* algorithm of $O(mn \log m)$ time.

Theorem 3. *There exists an inplace algorithm running in $O(mn \log m + occ)$ time for the combined problem of* RLE-WILDCARD *and* RLE-MISMATCH, *where occ is the number of the occurrences.*

Proof. The key is to observe that set S in ACCUMULATING is partially sorted, and thus sorting S is in fact merging $O(m)$ sorted lists. More specifically, suppose that $D_{i,j_1}, D_{i,j_2}, \ldots, D_{i,j_k}$, where $j_{k'} < j_{k'+1}$, are the mismatch blocks related to run X_i. We have that the turning points corresponding to the upper-left corners of $D_{i,j_1}, D_{i,j_1}, \ldots, D_{i,j_k}$, i.e. list $\langle y_{i,j_{k'}} - x_{i,j} \rangle_{1 \le k' \le k}$, are already in the right order. To retrieve them one-by-one, we need only traverse the runs of T_c once. The same property holds in the other three lists of turning points related to run X_i. Since there are m runs in P_c, the number of the sorted lists to be merged is at most $4m$. By adopting the heap-sort procedure, the intersections of the diagram of F can be computed on the fly in $O(mn \log m)$ time and $O(m)$ extra space. Finally, setting a threshold line to the diagram of F, we output the positions of all the occurrences on the fly. □

Corollary 2. *There exists an $O(mn \log m)$-time algorithm for the decision problems of* RLE-WILDCARD *and* RLE-MISMATCH.

Recall that Section 3 establishes the 3SUM-hardness for the decision problems of both RLE-WILDCARD and RLE-MISMATCH, suggesting an $\Omega(mn)$-time barrier. We post as an open problem whether the log-factor in Corollary 2 can be removed.

4.2. The function problems and sorting pairwise sums

The parameter *occ* in Theorem 3 may be of order $|T|$, independent of m and n. Below, we define *the function problem* of RLE-WILDCARD, denoted by RLE-WILDCARD-F, in such a way that the occurrences are reported in the RLE format.

Definition 5. The occurrence string Z of pattern P in text T is a bitstring of size $|T|$ such that $Z[i] = 1$ if P occurs at the text position i, and $Z[i] = 0$ otherwise. Let Z_c be the RLE compression of Z. We also say that Z_c is the RLE occurrence string of P in T.

Problem 5. RLE-WILDCARD-F: Assume that pattern P and text T, two strings over $\Sigma \cup \{*\}$, are compressed into the RLE strings P_c and T_c, respectively. The function problem of RLE-WILDCARD is, given P_c and T_c, to compute the RLE occurrence string Z_c of P in T.

Recall that the plane-sweep algorithm in Section 4.1 computes the $O(mn)$ intersections of the diagram of F. By setting a threshold line to F, one can easily compute the RLE occurrence string Z_c. As a side effect, we know that Z_c can be encoded as $O(mn)$ runs (since the diagram of F is composed of $O(mn)$ line segments, and setting a threshold line at most doubles the number).

Corollary 3. *There exists an inplace algorithm running in $O(mn \log m)$ time for* RLE-WILDCARD-F.

It is not hard to come up with an instance for which Z_c contains $\Theta(mn)$ runs. Thus, there exists an $\Omega(mn)$ trivial lower bound for RLE-WILDCARD-F. Below, we give a stronger result by showing that RLE-WILDCARD-F is at least as hard as the problem of sorting pairwise sums, denoted by $(x + y)$-SORTING, defined below.

Problem 6. $(x + y)$-SORTING: Given two sets of integers X and Y, each of size n, the problem is to sort the pairwise sums of X and Y, i.e. set $\{x + y \mid x \in X, y \in Y\}$.

$(x+y)$-SORTING is also a 3SUM-hard problem. Moreover, the obvious $\Theta(n^2 \log n)$-time solution has been the fastest known for a long time. Therefore, $(x + y)$-SORTING often serves as a base problem to indicate that some problems possess an $\Omega(n^2 \log n)$-time barrier. Analogously, we call a problem $(x + y)$-SORTING-hard if there is an $o(n^2 \log n)$ transformation from $(x + y)$-SORTING to that problem. Some of the 3SUM-hard problems are later recognized as $(x + y)$-SORTING-hard problems [6,13]. Below, we show that RLE-WILDCARD-F is $(x + y)$-SORTING-hard; and therefore, to improve the $O(mn \log m)$-time bound of Corollary 3, one had better improve the long-standing time complexity of $(x + y)$-SORTING first. (Similar results can be derived for the function problems of RLE-WILDCARD$_T$ and RLE-MISMATCH as well.)

Theorem 4. $(x + y)$-SORTING $\ll_{n^2}$ RLE-WILDCARD-F.

Proof. We will reduce sorting $Y - X$ to RLE-WILDCARD-F instead. Given X and Y, an instance of $(x+y)$-SORTING, without loss of generality we assume that $X \cup Y$ contains distinct positive numbers and that all the numbers in Y are greater than the numbers in X. By sorting X and Y, we can write $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, where $x_1 < x_2 < \cdots < x_n < y_1 < y_2 < \cdots < y_n$. We construct two RLE strings, P_c and T_c, as an instance of RLE-WILDCARD-F:

$$P_c = *^{x_1-1}\, 1^1\, *^{x_2-x_1-1}\, 1^1\, *^{x_3-x_2-1}\, 1^1 \ldots *^{x_n-x_{n-1}}\, 1^1,$$
$$T_c = 1^{y_1-1}0^1 1^{y_2-y_1-1}0^1 1^{y_3-y_2-1}0^1 \ldots 1^{y_n-y_{n-1}}0^1 1^{x_n-1}.$$

Observe that $Z[d] = 0$ if and only if there exists a pair x_i and y_j such that $y_j - x_i = d - 1$. Therefore, once we obtain the RLE occurrence string Z_c of P in T, it is easy to generate the sorted sequence of $Y - X$. □

4.3. Two-dimensional matching

In real applications, image data, as well as faxed documents, are often represented as two-dimensional arrays. The two-dimensional run-length encoding, used in fax transmission, is defined as the concatenation of the RLE compression of all rows (or columns). More specifically, let pattern P_{2d} and text T_{2d} be two rectangular arrays; we assume they are of sizes $k \times M$ and $\ell \times N$, respectively. The two-dimensional run-length encoding of P_{2d} is the RLE string $P_c = P_{c,1}P_{c,2} \dots P_{c,k}$, where $P_{c,i}$ is the RLE compression of row i of P_{2d}. Similarly, T_{2d} is compressed into the RLE string $T_c = T_{c,1}T_{c,2} \dots T_{c,\ell}$. If both P_{2d} and T_{2d} contain only one row, the problem becomes one-dimensional matching. Hence, the previously derived lower bounds hold for the two-dimensional matching problem. The two-dimensional problem can be transformed into a one-dimensional setting by the following standard technique. We insert a wildcard run $*^{N-M}$ into every two consecutive runs of P_c. One can see that, for $i \in [1, \ell - k + 1]$ and $j \in [1, N - M + 1]$, P_{2d} occurs at position (i, j) in T_{2d} if and only if P occurs at position $(i-1) \times N + j$ in T. Therefore, by imposing a position constraint on the output, our algorithm can be adapted to solve the two-dimensional matching problem. The time and space usage remain the same since the transformation only doubles the number of runs in P_c.

Theorem 5. *There exists an inplace algorithm running in $O(mn \log m + occ)$ time for the two-dimensional problem of* RLE-WILDCARD, RLE-MISMATCH, *or their combined problem, where m and n are the compressed sizes of the pattern and the text using two-dimensional run-length encoding. Note that the time complexity becomes $O(mn \log m)$ if their decision problem or their function problem, defined analogously to that of Section 4.2, is considered.*

5. Concluding remarks

In the weighted edit distance model, the distance (similarity) of two strings is measured by the minimum cost of transforming one string into the other via operations of substitutions, insertions, and deletions. The costs of these operations on different symbols are specified by a given scoring matrix. Given two strings of lengths M and N, compressed into m and n runs, Mäkinen et al. [18] and Crochemore et al. [10] independently proposed an upper bound of $O(mN + Mn)$ for the weighted edit distance model; a recent work improved the bound to $O(\min\{mN, Mn\})$ [14]. In this paper, we establish the 3SUM-hardness for the Hamming distance setting, in which a substitution of each symbol costs 1 and an insertion or deletion costs ∞. The 3SUM-hardness result extends naturally to the weighted edit distance model, since in that model the costs of operations can be arbitrarily assigned. However, the gap is large between the existing upper bounds and the $\Omega(mn)$ barrier, suggested by the 3SUM problem. Bridging the gap between the bounds remains open.

We close the paper by mentioning a few more open questions. As discussed at the end of Section 4.1, there exists a possible log-factor improvement for the decision problems of RLE-WILDCARD and RLE-MISMATCH. On the other hand, designing an algorithm for RLE-MISMATCH that is efficient for small K is interesting and practical. For example, an $O(Kn)$-time algorithm will outperform our solution when $K = o(m \log m)$. Moreover, it is also interesting to ask whether the 3SUM-hardness exists in other well-known distance settings, such as the LCS (longest common subsequence) metric and the Levenshtein distance. Both of them have drawn much attention in the field of pattern matching, and their corresponding compressed problems with respect to run-length encoding have also been extensively studied [5,16–19].

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments. Kuan-Yu Chen, Ping-Hui Hsu, and Kun-Mao Chao were supported in part by NSC grants 95-2221-E-002-126-MY3 and 97-2221-E-002-097-MY3 from the National Science Council, Taiwan.

References

[1] Karl Abrahamson, Generalized string matching, SIAM Journal on Computing 16 (6) (1987) 1039–1051.
[2] Amihood Amir, Gary Benson, Efficient two-dimensional compressed matching, DCC (1992) 279–288.

[3] Amihood Amir, Gad M. Landau, Dina Sokol, Inplace run-length 2d compressed search, Theoretical Computer Science 290 (3) (2003) 1361–1383.
[4] Amihood Amir, Moshe Lewenstein, Ely Porat, Faster algorithms for string matching with k mismatches, Journal of Algorithms 50 (2) (2004) 257–275.
[5] Alberto Apostolico, Gad M. Landau, Steven Skiena, Matching for run-length encoded strings, Journal of Complexity 15 (1) (1999) 4–16.
[6] Gill Barequet, Sariel Har-Peled, Polygon containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard, International Journal of Computational Geometry and Applications 11 (4) (2001) 465–474.
[7] Raphaël Clifford, Manolis Christodoulakis, Tim Crawford, David Meredith, Geraint A. Wiggins, A fast, randomised, maximal subset matching algorithm for document-level music retrieval, ISMIR (2006) 150–155.
[8] Peter Clifford, Raphaël Clifford, Simple deterministic wildcard matching, Information Processing Letters 101 (2) (2007) 53–54.
[9] Raphaël Clifford, Klim Efremenko, Ely Porat, Amir Rothschild, From coding theory to efficient pattern matching, SODA (2009) 778–784.
[10] Maxime Crochemore, Gad M. Landau, Michal Ziv-Ukelson, A subquadratic sequence alignment algorithm for unrestricted scoring matrices, SIAM Journal on Computing 32 (6) (2003) 1654–1673.
[11] Tali Eilam-Tsoreff, Uzi Vishkin, Matching patterns in a string subject to multilinear transformations, Theoretical Computer Science 60 (3) (1988) 231–254.
[12] Anka Gajentaan, Mark H. Overmars, On a class of $o(n^2)$ problems in computational geometry, Computational Geometry 5 (1995) 165–185.
[13] Antonio Hernández-Barrera, Finding an $o(n^2 \log n)$ algorithm is sometimes hard, CCCG (1996) 289–294.
[14] Guan-Shieng Huang, Jia Jie Liu, Yue-Li Wang, Sequence alignment algorithms for run-length-encoded strings, COCOON (2008) 319–330.
[15] Roy Hunter, A. Harry Robinson, International digital facsimile coding standards, Proceedings of the IEEE 68 (7) (1980) 854–867.
[16] Jia Jie Liu, Guan-Shieng Huang, Yue-Li Wang, Richard Chia-Tung Lee, Edit distance for a run-length-encoded string and an uncompressed string, Information Processing Letters 105 (1) (2007) 12–16.
[17] Jia Jie Liu, Yue-Li Wang, Richard Chia-Tung Lee, Finding a longest common subsequence between a run-length-encoded string and an uncompressed string, Journal of Complexity 24 (2) (2008) 173–184.
[18] Veli Mäkinen, Esko Ukkonen, Gonzalo Navarro, Approximate matching of run-length compressed strings, Algorithmica 35 (4) (2003) 347–369.
[19] Joseph S.B. Mitchell, A geometric shortest path problem, with application to computing a longest common subsequence in run-length encoded strings, Technical Report, SUNY Stony Brook, 1997.
[20] P.S.P. Wang, Handbook of Optical Character Recognition and Document Image Analysis, World Scientific Publishing Company, 1997.