Sequence Alignment Algorithms for Run-Length-Encoded Strings

Guan Shieng Huang[1,⋆], Jia Jie Liu[2], and Yue Li Wang[1,⋆⋆]

[1] Department of Computer Science and Information Engineering, National Chi Nan University, Taiwan
shieng@ncnu.edu.tw, yuelwang@ncnu.edu.tw
[2] Department of Information Management, Shih Hsin University, Taiwan
jjliu@cc.shu.edu.tw

Abstract. A unified framework is applied to solving various sequence comparison problems for run-length encoded strings. All of these algorithms take $O(\min\{mn', m'n\})$ time and $O(\max\{m, n\})$ space, for two strings of lengths m and n, with m' and n' runs, respectively. We assume the linear-gap model and make no assumption on the scoring matrices, which maximizes the applicability of these algorithms. The trace (i.e., the way to align two strings) of an optimal solution can also be recovered within the same time and space bounds.

1 Introduction

We consider how to compare the similarity of two run-length-compressed strings in this paper. Let Σ be an alphabet with a constant number of symbols. A string like a^k for any $a \in \Sigma$ and $k \in \mathbb{N}$ is called a *run*. Strings can be compressed by runs. For example, *aaaabbbaa* can be compressed to $(a, 4)(b, 3)(a, 2)$ as three runs in the run-length encoding. Let x and y be two strings over Σ with lengths m and n, respectively. Suppose x has m' runs and y has n' runs. In this paper, we show that the following problems can be solved in $O(m'n)$ time and $O(n)$ space under a unified framework: (A) the string edit distance problem; (B) the pairwise global alignment problem; and (C) the pairwise local alignment problem. We assume the linear-gap model, and the scoring matrices can be arbitrary.

Two special cases of (A) and (B) are the LCS (Longest-Common-Subsequence) metric and the Levenshtein metric [24,18]. For the LCS metric, Bunke and Csirik [8] first presented an $O(m'n + mn')$ time algorithm. It is further improved by Apostolico et al. [4] in time $O(m'n' \lg(m'n'))$, and by Mitchell [22] in time $O((m' + n' + d) \lg(m' + n' + d))$ where d is the number of matches of runs (in the worst case $d = O(m'n')$). Around 2002, three groups independently made a break through on this problem. Arbell et al. [5] solved the case for the Levenshtein metric, which is an edit distance problem with unit cost, in time $O(m'n + mn')$; Mäkinen et al. [21] solved the general edit distance problem,

⋆ Research supported in part by NSC 95-2221-E-260-016-MY2.
⋆⋆ Research supported in part by NSC 96-2221-E-260-018.

X. Hu and J. Wang (Eds.): COCOON 2008, LNCS 5092, pp. 319–330, 2008.

under the assumption that the distance matrix is nonnegative and satisfies the triangle inequality, in time $O(m'n + mn')$; Crochemore et al. [11] solved the global alignment problem under the linear-gap model, and the latter can be easily converted to solving the general edit distance problem, also in $O(m'n + mn')$ time. Finally, Liu et al. [19,20] proposed an $O(\min\{m'n, mn'\})$-time algorithm for the Levenshtein and LCS metrics.

The paper of Crochemore et al. [11] originally solves sequence alignment problems under the Lempel-Ziv encoding. Strings being compressed by LZ78 have nice recursive structures. They employed *the Monge property* (which will be explained in Sect. 3) to solve the global and the local alignment problems in both $O(hn^2/\lg n)$ time and space where $0 < h \leq 1$ is the entropy of input strings. The trace [24] can also be recovered in the same time and space bounds. This technique was also applied to run-length compressed strings for the global alignment problem. We remark that the approach of [11] has several advantages. First, its assumption is rather general. It assumes the linear-gap model with unrestricted scoring matrices. Second, this framework has great potential to solve other related problems such as the local alignment problem.

We combine techniques from [11] and [19], and improve several results of [21,11,19,20]. In addition, we propose an algorithm for the approximate matching under the run-length encoding. Let T be a long text, which is run-length-compressed into n' runs. Let P be a pattern with m characters. This problem asks one to find out all occurrences of P in T such that their distances are under some given threshold. Our algorithm takes $O(n'm)$ time and $O(m)$ space, which improves the previous $O(n'mm')$-time algorithm in [21]. Furthermore, we do not need the entries in a distance matrix and the threshold for approximate matching to be bounded by constants, which is implicitly assumed in the algorithm proposed by [21]. A comparison of related results is listed in Table 1.

Table 1. A comparison of related results

Problem	In This Paper	Previous Results
Edit distance/ Global alignment	$O(\min\{m'n, mn'\})$ time[a]	$O(m'n + mn')$ time [11][a] $O(\min\{m'n, mn'\})$ time[19,20][b]
Local alignment	$O(\min\{m'n, mn'\})$ time[a]	only for LZW compression [11][c]
Approximate matching	$O(n'm)$ time[a]	$O(n'mm')$ time [21][d]

[a] Linear gap cost with unrestricted scoring matrices.
[b] Limited to Levenshtein and LCS metrics.
[c] To the best of our knowledge.
[d] Assume the distance is a metric and the approximate threshold is a constant.

The organization of this paper is as follows. We explain the basic idea in Sect. 2 and give a quick review on related preliminaries in Sect. 3. Then subsequent sections are followed for each problem. Finally, a conclusion ends in Sect. 8.

2 The Idea

We integrate several important techniques from [11] and [19]. Let x be a run-length-compressed string which is put to the left side of the *edit graph* [12]. Let y be an uncompressed string which is put on the top. The edit graph is divided into several regions, and in each region, only the values on borders are computed. However, unlike [11,5,8,21] that partition the edit graph into blocks, we use strips. A strip R is a region defined by a run of x and the whole y (see Fig. 1). Let $O_R(j)$ be the jth cell in the last row of R. Let $I_R(i)$ be the ith cell in the last row of the region prior to R. We will show that, for each run of x, all values of O_R can be computed in $O(n)$ time based on values of I_R. Accordingly, the last row of the edit graph can be computed in overall $O(m'n)$ time.

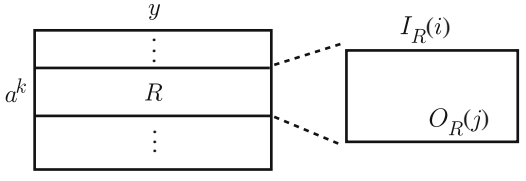

Fig. 1. A strip and its input/output borders

More specifically, let $DIST(i,j)$ be the cost of the optimal path starting from $I_R(i)$ and ending at $O_R(j)$ for $1 \le i \le j \le n$. Define $OUT(i,j) = I_R(i) + DIST(i,j)$. Hirschberg [13] observed[1] that

$$O_R(j) = \min_{1 \le i \le j} OUT(i,j) \quad \text{for } 1 \le j \le n \ . \tag{1}$$

The matrix OUT defines a *Monge* matrix [2]. Therefore, all of the minima in (1) for all j can be determined in $O(n)$ time by applying the SMAWK algorithm [1]. $DIST$ and OUT matrices have also been used elsewhere, see [11,3,7,14,16,23,17]. Related preliminaries are given in the next section.

3 Preliminaries

Definition 1. *An $m \times n$ matrix $M = (c_{i,j})_{m \times n}$ is called* Monge *iff*

$$c_{i,j} + c_{i',j'} \le c_{i,j'} + c_{i',j} \tag{2}$$

for all $1 \le i \le i' \le m$ and $1 \le j \le j' \le n$.

Symmetrically, we can define *an inverse Monge matrix* if (2) is replaced by $c_{i,j}+c_{i',j'} \ge c_{i,j'}+c_{i',j}$. Monge matrices have many nice properties and important applications. For surveys and recent applications, see [9,6,10].

Lemma 1 (Aggarwal and Park [2]). *All of the row minima and column minima in an $m \times n$ Monge or inverse Monge matrix can be determined in time*

[1] Hirschberg did not write in this form. It is borrowed from [11].

$O(m+n)$*, provided that each entry in the matrix can be accessed in time* $O(1)$*. (When there are many minima in a row or column, we can simply choose the first one.) Dually, all of the row and column maxima can also be found in the same time bound.*

The algorithm for Lemma 1, which is called the SMAWK algorithm, was first invented in [1] by applying the prune-and-search technique. Indeed, Aggarwal et al. in [1] defined a larger class called the *total monotonicity*. However, in real applications Monge or inverse Monge is sufficient for most cases. Extensions to higher dimensional arrays and on-line algorithms can be found in [2] and [6], respectively.

Lemma 2 (Aggarwal and Park [2]). *The matrices DIST and OUT used in (1) are Monge. When changing* min *to* max*, the corresponding DIST and OUT matrices are inverse Monge.*

In Appendix A, we give an informal proof of Lemma 2. As a consequence, if entries in $DIST$ can be accessed in $O(1)$ time, from Lemmas 1 and 2, all cells of O_R can be evaluated in $O(n)$ time for each strip R. In the following four sections, we will show how to define appropriate O_R, I_R, $DIST$, OUT for various problems related to sequence alignment under this framework. The key point is on how to keep $DIST(i,j)$ accessible in $O(1)$ time.

4 Edit Distance with Unrestricted Scoring Matrices

Let $E(x,y)$ be the edit distance of strings x and y in the linear-gap model with scoring matrix δ. It accounts for the minimum number of weighted edit operations (i.e., the insertions, deletions, and substitutions) that transform x into y.

The traditional way to compute the edit distance is through the following recurrence relation:

$$E(ua,vb) = \min\{E(ua,v)+\delta(-,b), E(u,vb)+\delta(a,-), E(u,v)+\delta(a,b)\} \quad (3)$$

with base cases $E(ua,\epsilon) = E(u,\epsilon)+\delta(a,-)$, $E(\epsilon,vb) = E(\epsilon,v)+\delta(-,b)$, and $E(\epsilon,\epsilon)=0$ where ϵ is the empty string and $-$ is the gap. We can replace (3) by

$$E(x'a^k, y[1..j]) = \min_{0\le i\le j}\left\{E(x', y[1..i]) + E(a^k, y[(i+1)..j])\right\} \quad (4)$$

when $x = x'a^k$ is run-length compressed and a^k is the last run of x. In fact, (4) is an instantiation of (1): set $I_R(i) = E(x', y[1..i])$ and $DIST(i,j) = E(a^k, y[(i+1)..j])$. All we have to do is to show that $E(a^k, y[(i+1)..j])$ can be evaluated in constant time, and consequently, the Monge paradigm described in Sects 2 and 3 ensures the $O(m'n)$ time and $O(n)$ space complexity.

For simplicity, let us first assume that the cost of an insertion or deletion is $d \ge 0$ and a substitution is $s \ge 0$. Then we have

Lemma 3. *Let* $a \in \Sigma$ *and* $z \in \Sigma^*$*. Let the length of* z *be* $|z|$ *and the number of occurrences of* a *in* z *be* $\sigma_a(z)$*. Then*

$$E(a^k, z) = d\max\{|z|, k\} - (d-s)\min\{|z|, k\} - s\min\{\sigma_a(z), k\} \quad \text{if } 0 \le s \le 2d \tag{5}$$

and

$$E(a^k, z) = d(|z| + k) - 2d\min\{\sigma_a(z), k\} \quad \text{if } s \ge 2d \ge 0\ . \tag{6}$$

Proof. The edit distance of two strings is $s\times\alpha + d\times(\beta+\gamma)$ where α, β, and γ are the numbers of substitutions, insertions, and deletions, respectively. Since the string a^k contains identical letters, these numbers can be easily counted. Suppose $s \le 2d$; then $\gamma+\beta = \max\{|z|, k\} - \min\{|z|, k\}$ and $\alpha = \min\{|z|, k\} - \min\{\sigma_a(z), k\}$. Hence (5) follows. Suppose $s \ge 2d$; then any substitution can be replaced by a deletion followed by an insertion, and thus, $\alpha = 0$ and $\beta + \gamma = |z| + k - 2\min\{\sigma_a(z), k\}$, the last term is the number of matches. Hence (6) follows.

We can spend $O(n)$ time to perform a linear scan on y in order to keep track of $\sigma_a(y[1..j])$ for $0 \le j \le n$, where $\sigma_a(y[1..0]) = \sigma_a(\epsilon) = 0$. Based on this preprocessing and the identity that $\sigma_a(y[(i+1)..j]) = \sigma_a(y[1..j]) - \sigma_a(y[1..i])$, Lemma 3 can be done in $O(1)$ time by instantiating z by $y[(i+1)..j]$.

Analysis of the time and space complexity. The preprocessing of $\sigma_a(y[1..j])$ for $0 \le j \le n$ takes $O(n)$ time and space, for each run in x. After this, each $DIST(i, j)$ and $OUT(i, j)$ can be accessed in $O(1)$ for $1 \le i \le j \le n$. Then applying the SMAWK algorithm on OUT matrix to find all of the column minima takes again $O(n)$ time and space. Note that the SMAWK algorithm need not evaluate all entries of the OUT matrix; it only scans $O(n)$ of them. Therefore, by using $O(n)$ time and space, we can advance the computation of the edit graph by a run of x. Hence the overall computation of $E(x, y)$ takes $O(m'n)$ time and $O(n)$ space. The time can be reduced to $O(\min\{m'n, mn'\})$ by choosing $E(x, y)$ or $E(y, x)$ according to which one costs less, and the corresponding space is $O(n)$ and $O(m)$, respectively.

As for a general scoring matrix, Lemma 3 can be extended smoothly when the size of the alphabet Σ is bounded. Let us reconsider the settings in Lemma 3, except now the weight for each edit operation is determined by δ. The following greedy algorithm can evaluate $E(a^k, y[(i+1)..j])$ for each i and j in $O(1)$ time. The evaluation of $E(a^k, z)$ inquires either to assign every a in a^k to a character in z or to delete itself. For those unmapped characters in z, insertions are applied. Define $\delta'(a, b) = \min\{\delta(a, b), \delta(a, -) + \delta(-, b)\}$ for all $b \in \Sigma$ (here a and b may be the same character). That is, if a substitution cannot be better than a deletion followed by an insertion, we never use it. Since the alphabet is bounded, without loss of generality, we can assume $\delta'(a, b_1) \le \delta'(a, b_2) \le \cdots \le \delta'(a, b_{|\Sigma|})$ for all $b_t \in \Sigma$. Hence this provides a priority to choose the mate for each a in a^k to z by $b_1 \prec b_2 \prec \cdots \prec b_{|\Sigma|} \prec -$. Let the occurrences of b_t in z be $\sigma_{b_t}(z)$. Also, set $\sigma_-(z)$ be k, which serves as the sentinel. Hence there exists u between 1 and $|\Sigma|$ such that $\sum_{1\le t\le u} \sigma_{b_t}(z) \ge k$ but $\sum_{1\le t\le u-1} \sigma_{b_t}(z) < k$ (here $b_{|\Sigma|+1} = -$). Let $A + B = \sigma_{b_u}(z)$ such that $A + \sum_{1\le t\le u-1} \sigma_{b_t}(z) = k$. Then the value of $E(a^k, z)$ equals to

$$A \cdot \delta'(a, b_u) + B \cdot \delta'(-, b_u) + \sum_{1 \le t \le u-1} \delta'(a, b_t) \cdot \sigma_{b_t}(z) + \sum_{u+1 \le t \le |\Sigma|} \delta'(-, b_t) \cdot \sigma_{b_t}(z) \ ,$$

which generalizes Lemma 3 and can be computed in $O(u) = O(|\Sigma|) = O(1)$ time.

Theorem 1. *The edit distance problem in the linear-gap model with unrestricted scoring matrices can be solved in* $O(\min\{mn', m'n\})$ *time for run-length-encoded strings* x *and* y *with lengths* m *and* n*, being compressed into* m' *and* n' *runs, respectively.*

5 Global Alignment Algorithm

The algorithm for the global alignment problem is simply the dual of the one described in Sect. 4. In this case, (1) is modified into

$$O_R(j) = \max_{1 \le i \le j} OUT(i, j) \quad \text{for } 1 \le j \le n \ ,$$

and the inverse Monge property holds for the OUT matrix. Each entry in the OUT matrix can be accessed in $O(1)$ time by a similar greedy algorithm for the edit distance problem, as long as the alphabet is bounded. Then all of the column maxima, which are the values of O_R, can be found by the SMAWK algorithm in $O(n)$ time and space. This process can be continued, and finally the value of the optimal global alignment of x and y can be obtained in $O(m'n)$ time and $O(n)$ space. The trace of an optimal alignment can also be found in the same time and space bounds, by applying Hirschberg's technique [13].

Theorem 2. *The global alignment problem in the linear-gap model with unrestricted scoring matrices can be solved in* $O(\min\{mn', m'n\})$ *time for run-length-encoded strings* x *and* y *with lengths* m *and* n*, being compressed into* m' *and* n' *runs, respectively.*

6 Local Alignment Algorithm

In the local alignment problem, the goal is to identify a substring x' of x and a substring y' of y such that the global alignment score of x' and y' is maximized. Let $L(x, y)$ denote the score of the optimal local alignment for x and y ending at the ends of x and y. The traditional approach evaluates $L(x, y)$ through the following recurrence relation:

$$L(ua, vb) = \max\{0, L(ua, v) + \delta(-, b), L(u, vb) + \delta(a, -), L(u, v) + \delta(a, b)\} \quad (7)$$

with base cases $L(ua, \epsilon) = \max\{0, L(u, \epsilon) + \delta(a, -)\}$, $L(\epsilon, vb) = \max\{0, L(\epsilon, v) + \delta(-, b)\}$, and $L(\epsilon, \epsilon) = 0$. The score of the optimal local alignment for x and y can be obtained by finding the maximum over all cells in the induced alignment graph.

There are two kinds of local alignment paths in the alignment graph. The first kind categorizes paths fully contained in a strip. The second kind has paths that

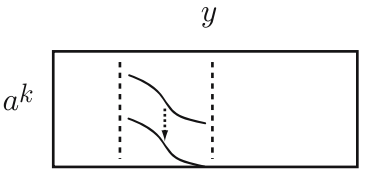

Fig. 2. An alignment inside a strip can always be moved down (or up) to touch the border in a local alignment graph

cross at least one border of a strip. Since the left side of a strip is always a whole run of x, paths of the first kind in fact can be *moved down or up* to touch the bottom or top of the strip (see Fig. 2 for an illustration). Hence it has at least one incarnation of the second kind, and thus can be ignored during the computation.

Paths of the second kind have a general pattern. Each path has three parts: S, H, E, which stand for segments of the path as follows (see Fig. 3):

S : a segment begins at some strip and ends at the bottom of this strip;
H : a segment begins at the top of a strip and ends at the bottom of some strip;
E : a segment begins at the top of a strip and ends within this strip.

Each part may be empty. The H part can be handled just like paths in the global alignment problem in a way propagating from one strip to the next. We can first ignore E part since their values cannot propagate to other strips, and after finishing values on borders of strips in the alignment graph, we can regain the contributions of E part into the alignment graph. As to incorporate S part into the alignment graph, we need to modify (4) into

$$L(x'a^k, y[1..j]) = \max\left\{\max_{0\le i\le j}\left\{L(x', y[1..i]) + G(a^k, y[(i+1)..j])\right\}, L(a^k, y[1..j])\right\} \tag{8}$$

where $G(a^k, y[(i+1)..j]$ is the *global* alignment score that can be evaluated in O(1) time. Note that (8) is different from (4) in the last term $L(a^k, y[1..j])$. Hence we create an array $S_R(j)$ for $0 \le j \le n$ at O_R in order to evaluate (8) where $S_R(j) = L(a^k, y[1..j])$. In the following paragraphs, we show how to calculate all entries of S_R in O(n) time. Again, the trick is the inverse Monge property.

Let us reformulate the framework provided in Sect. 2 for the local alignment problem. The meanings of I_R and O_R are the same, but at this time they record the scores of optimal local alignments ending at the corresponding cells. In other words, $I_R(i) = L(x', y[1..i])$ and $O_R(j) = L(x'a^k, y[1..j])$. Leave $DIST(i,j) = G(a^k, y[(i+1)..j])$ and $OUT(i,j) = I_R(i) + DIST(i,j)$ unchanged. Then (8) can be rewritten into

$$O_R(j) = \max\left\{\max_{0\le i\le j} OUT(i,j), S_R(j)\right\} .$$

The computation of $\max_{0\le i\le j} OUT(i,j)$ for $0 \le j \le n$ is as before, and it can be finished in O(n) time and space, and thus the H part follows.

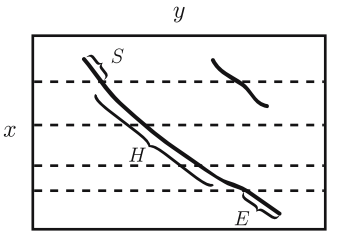

Fig. 3. Paths need to be considered in a local alignment graph

The calculations of S_R are closely related to $\max_{0 \le i \le j} DIST(i,j)$ for $0 \le j \le n$. Given j, let $i(j)$ be the first i that maximizes $DIST(i,j)$. It stands for the score of an optimal *local alignment* ending at $O_R(j)$ under the constraint that it must use exactly k copies of a's from a^k, a run of x. However, $S_R(j)$ stands for the optimal local alignment ending at $O_R(j)$ that uses *at most* k copies of a's. This difference can be resolved by applying this simple idea: set $\delta(a,-)$ to be zero; that is, the deletion of a's from a^k is free so that 'exactly' performs as well as 'at most'. This trick can be applied because we can always regroup all of the deletions of a's to its very beginning. As a consequence they can be truncated if deleting an a causes a negative score. By the above discussion, we have to consider two cases: (1) $\delta(a,-) < 0$ and (2) $\delta(a,-) \ge 0$. For the first case, let $\delta_{\mathrm{L}}(a,-) = 0$ and keep all other scores the same as δ. Let $DIST_{\mathrm{L}}(i,j)$ be the score of the optimal global alignment of a^k against $y[(i+1)..j]$ under δ_{L}. Then $DIST_{\mathrm{L}}$ is inverse Monge and the resulting $\max_i DIST_{\mathrm{L}}(i,j)$, which is the required $S_R(j)$, for $0 \le j \le n$, can be computed in $\mathrm{O}(n)$ time and space. The second case deals with $\delta(a,-) \ge 0$ (this could happen since δ is arbitrary). This case is much easier since $L(a^k, y[(i+1)..j])$ always uses all a^k and thus, $L(a^k, y[(i+1)..j])$ coincides with $DIST(i,j)$. Therefore $S_R(j) = \max_i DIST(i,j)$.

So far, we know how to handle S and H parts in the local alignment graph. In order to handle the E part, we need to create another array $E_R(i)$ for $0 \le i \le n$ at I_R such that $E_R(i) = L(a^k, y[(i+1)..n])$. This case can be processed similarly as the procedure for the S part.

The final optimal local alignment for $L(x,y)$ can be found by a linear scan over all entries in the output border O_R plus the corresponding contributions from the E part.

Analysis of the time and space complexity. It is easy to see that the time and space complexity is the same as the algorithm for the global alignment problem. The trace of an optimal local alignment can also be constructed in $\mathrm{O}(m'n)$ time and $\mathrm{O}(n)$ space by Hirschberg's technique [13].

7 Approximate Matchings

Given a text T, a pattern P, a distance matrix δ, and a bound K, the problem of approximate matching is to find all occurrences of substrings T' of T such

that the edit distance of T' and P under δ does not exceed the bound K. Its dual is to find out all occurrences of substrings whose global alignment scores against P are above the bound K.

Let T be run-length-encoded, with length n and n' runs. Let the length of P be m. We first propose an $O(mn')$ time and $O(m)$ space algorithm that can locate all end positions indexed by runs. The exact end locations of T' on T can further be reported as intervals on T in the same time and space bound. Consequently, all of the r occurrences can be reported with additional $O(r)$ time.

The idea is very similar to the algorithm in Sect. 6. We simply sketch the idea here. We put P on top of the edit graph and T to the left. Again, each optimal path can be broken into three parts: S, H, and E. However, in this time the prefixes and suffixes of P (which corresponds to y) cannot be truncated, but this facility is still available on T. The only difference is to set $S_R(j) = DIST_{\mathrm{L}}(0, j)$ instead of $\min_i DIST_{\mathrm{L}}(i, j)$. This prohibits the possibility to truncate the prefix of P. The E part is processed similarly, which prevents the removal of suffixes of P. If there exists a strip R and an index j such that $O_R(j) + E_{R'}(j)$ below the threshold K, where R' is the strip next to R, then we find an occurrence.

The next step is on how to locate all end positions of T' on T. Let T' be such an occurrence, and let R the the strip that contains the E part of T'. Let $I_R(i)$ be the starting position of E for T'. Since $I_R(i)$ records the distance of the optimal trace among all paths passing through it, its value cannot be worse than T'. Hence if we can find all segments, starting from $I_R(i)$ and ending to the right border of R, such that each of their edit distance against $P[(i+1)..m]$ plus the value of $I_R(i)$ is below K, then this is a valid occurrence. The greedy algorithm proposed in the end of Sect. 4 can be modified to fulfil this requirement. For each $I_R(i)$, the corresponding end locations of these segments (which are on the right border of R) then can be represented by an interval. (This is because we regroup letters in $P[(i+1)..m]$ into $|\Sigma|$ buckets, and the cost of mating or deleting one a to the same bucket costs the same, and the costs for buckets are sorted.) An additional check can also report approximate matches that occurred inside a strip.

Theorem 3. *Let T be a text with length n, being run-length-compressed into n' runs. Let P be a pattern with m characters. For any K, which serves as the threshold for matching, all of the occurrences T' of T such that the edit distance of T' and P is bounded above by K can be determined in $O(mn')$ time and $O(m)$ space. The threshold K and entries in the distance matrix δ can depend on m and n'.*

8 Conclusion

We improve previous $O(m'n + nm')$ time algorithms to $O(m'n)$, which makes one of the string truly compressed in the run-length encoding. However, another string is still left flat. An intriguing question is on (if it is possible) how to design an efficient algorithm for each problem whose time complexity only depends on the numbers of runs, as what were done in [4,22] for the LCS problem.

Another direction to extend the applicability of these algorithms is to introduce the *affine gap penalty*, which takes consecutive gaps as a whole unit and the costs for opening and extending a gap are different. In [15], Kim et al. proposed an $O(m'n + mn')$-time algorithm for the global alignment problem with affine gap penalty on run-length-encoded strings. It uses similar techniques as in [5,21]. However, Ledergerber et al. in [17] have pointed out that the matrix $OUT(i, j)$ described in Sect. 2 might not be (inverse) Monge. Therefore direct application of the model provided in Sect. 2 seems impossible. We note that it is possible to combine the results of [19] and [15] to get an $O(\min\{mn', m'n\})$-time algorithm for the affine gap penalty.

References

1. Aggarwal, A., Klawe, M.M., Moran, S., Shor, P., Wilher, R.: Geometric Applications of a Matrix-Searching Algorithm. Algorithmica 2(1), 195–208 (1987)
2. Aggarwal, A., Park, J.: Notes on Searching in Multidimensional Monotone Arrays. In: Proceedings of the 29th IEEE Symposium on Foundations of Computer Science (FOCS 1988), pp. 497–512 (1988)
3. Apostolico, A., Atallah, M.J., Larmore, L.L., Mcfaddin, S.: Efficient Parallel Algorithms for String Editing and Related Problems. SIAM Journal on Computing 19(5), 968–988 (1990)
4. Apostolico, A., Landau, G.M., Skiena, S.: Matching for Run-Length Encoded Strings. Journal of Complexity 15(1), 4–16 (1999)
5. Arbell, O., Landau, G.M., Mitchell, J.S.B.: Edit Distance of Run-Length Encoded Strings. Information Processing Letters 83(6), 307–314 (2002)
6. Bein, W.W., Golin, M.J., Larmore, L.L., Zhang, Y.: The Knuth-Yao Quadrangle-Inequality Speedup is a Consequence of Total-Monotonicity. In: Proceedings of the 7th annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006), pp. 31–40 (2006)
7. Benson, G.: A Space Efficient Algorithm for Finding the Best Nonoverlapping Alignment Score. Theoretical Computer Science 145(1–2), 357–369 (1995)
8. Bunke, H., Csirik, J.: An Improved Algorithm for Computing the Edit Distance of Run-Length Coded Strings. Information Processing Letters 54(2), 93–96 (1995)
9. Burkard, R.E., Klinz, B., Rudolf, R.: Perspectives of Monge Properties in Optimization. Discrete Applied Mathematics 70(2), 95–161 (1996)
10. Burkard, R.E.: Monge Properties, Discrete Convexity and Applications. European Journal of Operational Research 176(1), 1–14 (2007)
11. Crochemore, M., Landau, G.M., Ziv-Ukelson, M.: A Subquadratic Sequence Alignment Algorithm for Unrestricted Scoring Matrices. SIAM Journal on Computing 32(6), 1654–1673 (2003)
12. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, Cambridge (1997)
13. Hirschberg, D.S.: A Linear Space Algorithm for Computing Maximal Common Subsequences. Communications of the ACM 18(6), 341–343 (1975)
14. Kannan, S.K., Myers, E.W.: An Algorithm for Locating Nonoverlapping Regions of Maximum Alignment Score. SIAM Journal on Computing 25(3), 648–662 (1996)
15. Kim, J.W., Amir, A., Landau, G.M., Park, K.: Computing Similarity of Run-Length Encoded Strings with Affine Gap Penalty. In: Consens, M.P., Navarro, G. (eds.) SPIRE 2005. LNCS, vol. 3772, pp. 315–326. Springer, Heidelberg (2005)

16. Landau, G.M., Ziv-Ukelson, M.: On the Common Substring Alignment Problem. Journal of Algorithms 41(2), 338–359 (2001)
17. Ledergerber, C., Dessimoz, C.: Alignments with Non-overlapping Moves, Inversions and Tandem Duplications in $o(n^4)$ Time. Journal of Combinatorial Optimization (to appear, 2007)
18. Levenshtein, V.I.: Binary Codes Capable of Correcting, Deletions, Insertions and Reversals. Soviet Physics Doklady 10, 707–710 (1966)
19. Liu, J.J., Huang, G.S., Wang, Y.L., Lee, R.C.T.: Edit Distance for a Run-Length-Encoded String and an Uncompressed String. Information Processing Letters 105(1), 12–16 (2007)
20. Liu, J.J., Wang, Y.L., Lee, R.C.T.: Finding a Longest Common Subsequence Between a Run-Length-Encoded String and an Uncompressed String. Journal of Complexity (to appear, 2008)
21. Mäkinen, V., Navarro, G., Ukkonen, E.: Approximate Matching of Run-Length Compressed Strings. Algorithmica 35(4), 347–369 (2003)
22. Mitchell, J.: A Geometric Shortest Path Problem, with Application to Computing a Longest Common Subsequence in Run-Length Encoded Strings. Technical report, SUNY Stony Brook (1997)
23. Schmidt, J.P.: All Highest Scoring Paths in Weighted Grid Graphs and Their Application to Finding All Approximate Repeats in Strings. SIAM Journal on Computing 27(4), 972–992 (1998)
24. Wagner, R.A., Fischer, M.J.: The String-to-String Correction Problem. Journal of the ACM 21(1), 168–173 (1974)

A Monge Property

The reason that OUT matrix in (1) is Monge (or inverse Monge) is as follows. As for minimization problems, such as the edit distance problem, we intend to find *shortest paths* in edit graphs. First, observe that $OUT(i, j)$'s are defined for $1 \leq i \leq j \leq n$, whose domain is not a square (see Fig. 4). Let us first consider $1 \leq i < i' \leq j < j' \leq n$. In Fig. 5, $DIST(i, j)$ and $DIST(i', j')$ are the lengths of shortest paths (drawn solid) from i to j and from i' to j', respectively. Similarly, the dashed lines are the shortest paths for i-j' and i'-j. Intuitively, $DIST(i, j) \leq DIST(i, c) + DIST(c, j)$ and $DIST(i', j') \leq DIST(i', c) + DIST(c, j')$, as long as lengths of paths are additive in the edit graph. This holds because $DIST(i, j)$ is the length of a shortest path connecting i and j, and $DIST(i, c) + DIST(c, j)$ represents the length of *another* path which may not be the shortest. We remark that this result does not rely on the condition of the triangle inequality for weights on edges; in fact, triangle inequality may not hold for paths on an edit graph. On the other hand, we have $DIST(i, j') + DIST(i', j) = DIST(i, c) + DIST(c, j') + DIST(i', c) + DIST(c, j)$. Combining them, we get

$$DIST(i, j) + DIST(i', j') \leq DIST(i, j') + DIST(i', j) \ ,$$

and thus

$$OUT(i, j) + OUT(i', j') \leq OUT(i, j') + OUT(i', j)$$

for $1 \leq i < i' \leq j < j' \leq n$. The above argument partially fulfils the Monge property. As for $1 \leq j < i \leq n$, this can be resolved by setting $OUT(i, j) = \infty$

and taking the convention that $r + \infty = \infty + r = \infty$ and $\infty + \infty = \infty$ for any real number r. Hence the matrix $OUT(i, j)$ for $1 \leq i, j \leq n$ in the minimization version is Monge.

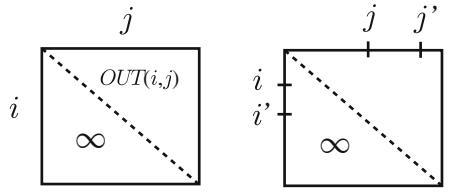

Fig. 4. Extension of the OUT matrix to the Monge matrix

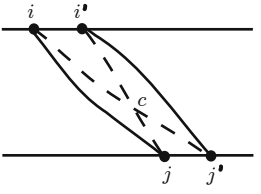

Fig. 5. A geometric interpretation of the quadrangle inequality

As for maximization problems, such as sequence alignment problems, we intend to find *longest paths* in the *alignment* graphs (note that an alignment graph is always acyclic). The argument in the above paragraph can also be applied by setting $OUT(i, j) = -\infty$ for $1 \leq j < i \leq n$ and taking the convention that $r + -\infty = -\infty + r = -\infty$ and $-\infty + -\infty = -\infty$ for any real number r. Finally, we conclude that the matrix $OUT(i, j)$ for $1 \leq i, j \leq n$ in the maximization version is inverse Monge.