Fast Algorithms for Computing the Constrained LCS of Run-Length Encoded Strings

Hsing-Yen Ann, Chang-Biau Yang†, Chiou-Ting Tseng and Chiou-Yi Hor
Department of Computer Science and Engineering,
National Sun Yat-sen University, Kaohsiung 80424, Taiwan
†Corresponding author: cbyang@cse.nsysu.edu.tw

Abstract— *In the constrained longest common subsequence (CLCS) problem, we are given two sequences X, Y and the constrained sequence P in run-length encoded (RLE) format, where $|X| = n$, $|Y| = m$ and $|P| = r$ and the numbers of runs in RLE format are N, M and R, respectively. In this paper, we show that after the sequences are encoded, the CLCS problem can be solved in $O(NMr + r \times \min\{q_1, q_2\} + q_3)$ time, where q_1 and q_2 denote the numbers of elements in the bottom and right boundaries of the partially matched blocks on the first layer, and q_3 denotes the number of elements of whole boundaries of all fully matched cuboids in the DP lattice. If the compression ratio is good, our work obviously outperforms the previously known DP algorithm and the Hunt-and-Szymanski-like algorithm.*

1. Introduction

The *longest common subsequence* (LCS) problem is a well-known measurement for computing the similarity of two strings. It can be widely applied in diverse areas, such as file comparison, pattern matching and computational biology. The most referred algorithm to solve the LCS is proposed by Wagner and Fischer [1]. Other advanced algorithms can also be found in some papers [2], [3], [4], [5].

Run-length encoding (RLE) is a well-known and simple method for compressing strings [6]. It divides a string into a sequence of *runs* where each run is a maximal repetitive substring of an identical symbol and can be represented as a pair, the symbol and the length. For example, a string $aaaddccccbbbbbb$ is encoded as $a^3d^2c^4b^6$ in the RLE format.

In our previous work [7], we considered the LCS problem for RLE strings and proposed an efficient algorithm, based on the *dynamic programming* (DP) technique, which outperforms the previous works [8], [9], [10]. In this paper, we consider the *constrained LCS* (CLCS) problem, which was first addressed by Tsai [11]. We will solve it by using the idea similar to our previous work [7].

Given two input sequences X, Y and a constrained sequence P, the CLCS problem is to find the longest common subsequences Z of X and Y such that P is a subsequence of Z. Tsai also proposed an algorithm to solve this problem in $O(n^2m^2r)$ time, where $|X| = n$, $|Y| = m$ and $|P| = r$. Two improved algorithms [12], [13] based on the DP technique were presented independently for solving this problem with $O(nmr)$ time and space complexity. Recently, Iliopoulos and Rahman [14] proposed a Hunt-and-Szymanski-like algorithm [3] and employed a special data structure, van Emde Boas tree [15], to solve this problem in $O(r\mathcal{R}\log\log(n+m))$ time, where $\mathcal{R}$ is the total number of ordered pairs of positions at which the two strings match. The Hunt-and-Szymanski-like algorithms are very efficient when $\mathcal{R}$ is small. However, it should be noted that $\mathcal{R} = O(nmr)$ in the worst case.

In this paper, we consider how to compute the CLCS of the strings which are in RLE format. By extending the concepts of the *blocks* [8], [16] and our previous work [7], we propose an algorithm to solve the CLCS problem in $O(NMr+r\times\min\{q_1, q_2\}+q_3)$ time, where q_1 and q_2 denote the numbers of elements in the bottom and right boundaries of the partially matched blocks on the first layer, and q_3 denotes the number of elements of whole boundaries of all fully matched cuboids in the DP lattice.

2. Preliminaries and Properties

Let X, Y denote the two input strings and P denotes the constrained sequence, where $X = x_1x_2\cdots x_n$, $Y = y_1y_2\cdots y_m$ and $P = p_1p_2\cdots p_r$ over a finite alphabet Σ. The lengths of X, Y and P are denoted as n, m and r, respectively. If a string is of zero length, it is denoted by an empty string ϕ. The string X is run-length-encoded into N runs, $RX_1, RX_2, \cdots, RX_N$, where the lengths of the runs are denoted by $n_1, n_2, \cdots, n_N$, respectively. Similarly, the RLE strings of Y and P are encoded as $RY_1RY_2\cdots RY_M$ and $RP_1RP_2\cdots RP_R$, whose lengths are denoted by $m_1, m_2, \cdots, m_M$ and $r_1, r_2, \cdots, r_R$, respectively. A substring $x_ix_{i+1}\cdots x_j$ of X is denoted as $X_{i..j}$. The consecutive runs $RX_i, RX_{i+1}, \cdots, RX_j$ is denoted as $RX_{i..j}$, which is also a substring of X. We denote the longest common subsequence Z of X and Y as $LCS(X, Y)$, whose length is denoted as $\mathcal{L}(X, Y)$. We also denote the CLCS of X and Y with respect to P as $CLCS(X, Y, P)$, whose length is denoted as $\mathcal{CL}(X, Y, P)$.

Lemma 1: (LCS). (See [1].) Given two strings X and Y and two distinct symbols a and b, the following conditions hold:

1) $\mathcal{L}(X, \phi) = 0$.
2) $\mathcal{L}(\phi, Y) = 0$.
3) $\mathcal{L}(Xa, Ya) = \mathcal{L}(X, Y) + 1$.
4) $\mathcal{L}(Xa, Yb) = \max\{\mathcal{L}(Xa, Y), \mathcal{L}(X, Yb)\}$.

Lemma 2: (Constrained LCS). (See [13].) Given two input strings X and Y, a constrained sequence P, and two distinct symbols a and b, the following conditions hold:

1) $\mathcal{CL}(\phi, Y, P) = -\infty$ if $P \neq \phi$.
2) $\mathcal{CL}(X, \phi, P) = -\infty$ if $P \neq \phi$.
3) $\mathcal{CL}(X, Y, \phi) = \mathcal{L}(X, Y)$.
4) $\mathcal{CL}(Xa, Ya, Pa) = \mathcal{CL}(X, Y, P) + 1$.
5) $\mathcal{CL}(Xa, Ya, Pb) = \mathcal{CL}(X, Y, Pb) + 1$.
6) $\mathcal{CL}(Xa, Yb, P) = \max\{\mathcal{CL}(Xa, Y, P), \mathcal{CL}(X, Yb, P)\}$.

Figure 1 shows an example of the CLCS problem, where $X = a^3b^6c^4a^{12}$, $Y = b^3a^8c^4a^5b^8c^4a^4$ and $P = ab$. We can see that, in the lowest layer, L_0, the operations of the DP lattice are the same to the LCS problem, however, things change in the higher layers.

Lemma 3: (LCS for RLE). (See [7], [16].) Given two strings X and Y and two different symbols a and b, the following conditions hold:

1) $\mathcal{L}(Xa^s, Ya^s) = \mathcal{L}(X, Y) + s$.
2) $\mathcal{L}(Xa^{s_1}, Ya^{s_2}) = \mathcal{L}(Xa^{s_1-s}, Ya^{s_2-s}) + s$, where $s = \min\{s_1, s_2\}$.
3) $\mathcal{L}(Xa^s, Yb^t) = \max\{\mathcal{L}(Xa^s, Y), \mathcal{L}(X, Yb^t)\}$.
4) (Merged white blocks). $\mathcal{L}(Xa_1^{s_1}a_2^{s_2}\cdots a_i^{s_i}, Yb_1^{t_1}b_2^{t_2}\cdots b_j^{t_j}) = \max\{\mathcal{L}(X, Yb_1^{t_1}b_2^{t_2}\cdots b_j^{t_j}), \mathcal{L}(Xa_1^{s_1}a_2^{s_2}\cdots a_i^{s_i}, Y)\}$, where $a_{i'} \neq b_{j'}$ for each $i' \in [1, i]$ and $j' \in [1, j]$.

According to the facts shown in Lemma 3, the concept that splits the original DP lattice into *blocks* was proposed [8], where each block corresponds to a pair of runs of X and Y. If the symbol of run RX_i is identical to the symbol of run RY_j, we say that the block $B_{i,j}$ is a *matched* (gray) block, otherwise it is a *mismatched* (white) block. The example of dividing the DP lattice of the LCS problem can be found in Figure 1(a) .

By applying the concept of block splitting, $\mathcal{L}(X, Y)$ can be computed according to the elements on the boundaries of the blocks, rather than the whole DP lattice. In other words, the value of each element inside the blocks is never evaluated, as shown in Figure 2(a). Liu et al. proposed the concept to process the elements of the DP lattice run by run rather than row by row [10], and the number of computed elements are further reduced, as shown in Figure 2(b). The number of elements to be computed in our previous algorithm [7] are further reduced, as shown in Figure 2(c).

3. Our Algorithms

Some additional properties of the CLCS problem for RLE strings can be easily deduced by combining Lemma 2 and Lemma 3 as follows.

Corollary 1: (Constrained LCS of RLE strings). Given two input strings X and Y, a constrained sequence P, and two distinct symbols a and b, the following conditions hold:

1) $\mathcal{CL}(Xa^{s_1}, Ya^{s_2}, Pa^{s_3}) = \mathcal{CL}(Xa^{s_1-s}, Ya^{s_2-s}, Pa^{s_3-s}) + s$, where $s = \min\{s_1, s_2, s_3\}$.
2) $\mathcal{CL}(Xa^{s_1}, Ya^{s_2}, Pb^t) = \mathcal{CL}(Xa^{s_1-s}, Ya^{s_2-s}, Pb^t) + s$, where $s = \min\{s_1, s_2\}$.
3) $\mathcal{CL}(Xa^s, Yb^t, P) = \max\{\mathcal{CL}(Xa^s, Y, P), \mathcal{CL}(X, Yb^t, P)\}$.
4) (Merged white blocks). $\mathcal{CL}(Xa_1^{s_1}a_2^{s_2}\cdots a_i^{s_i}, Yb_1^{t_1}b_2^{t_2}\cdots b_j^{t_j}, P) = \max\{\mathcal{CL}(Xa_1^{s_1}a_2^{s_2}\cdots a_i^{s_i}, Y, P), \mathcal{CL}(X, Yb_1^{t_1}b_2^{t_2}\cdots b_j^{t_j}, P)\}$, where $a_{i'} \neq b_{j'}$ for each $i' \in [1, i]$ and $j' \in [1, j]$.

According to Corollary 1, we can apply the concept of block splitting proposed by Bunke and Csirik [8]. Therefore, in our new algorithm, a 3-dimensional DP lattice is divided into $N \times M \times R$ *cuboids*, where each cuboid corresponds to a tuple of runs of X, Y and P.

As shown in Figure 1, if only one layer of the DP lattice is considered, the diagonal pattern blocks, gray blocks and white blocks represent the slices of the fully matched cuboids, partially matched cuboids and mismatched cuboids, respectively. The behaviors of the gray blocks and white blocks of Figure 1 are the same as the matched blocks and mismatched blocks in Bunke and Csirik's algorithm [8].

We should also note that, only the boundary elements of each cuboid of the DP lattice have to be evaluated. This means that the evaluated elements of each cuboid form a hollow cuboid. It is clear that, under the *random access machine* (RAM) model, we can easily apply the technique proposed by Bunke and Csirik [8] to prevent the allocation and initialization of the elements which are never evaluated.

By observing the dependencies of the elements in the DP lattice, the following facts can be found.

Lemma 4: (Two kinds of paths). Given a layer of the DP lattice for the CLCS problem for RLE strings, the following conditions hold:

1) If the symbol of the current run RX_i in X is different from the current symbol in P, the blocks corresponding to the current run RX_i consist of partially matched blocks and mismatched blocks.
2) If the symbol of the current run RX_i is identical to the current symbol in P, the blocks corresponding to the current run RX_i consist of fully matched blocks and mismatched blocks.

We can see that the optimal RMQ technique used in our previous work [7] still works correctly when this kind of runs are considered. However, an element corresponds to an element on the lower layer if it is located in a fully matched block (or cuboid). This breaks the rule of our previous work [7], and thus we can only apply the property of *merged white blocks*. In summary, we get the following theorem.

Theorem 1: Given two input sequences X, Y and a constrained sequence P with lengths n, m and r, respectively,

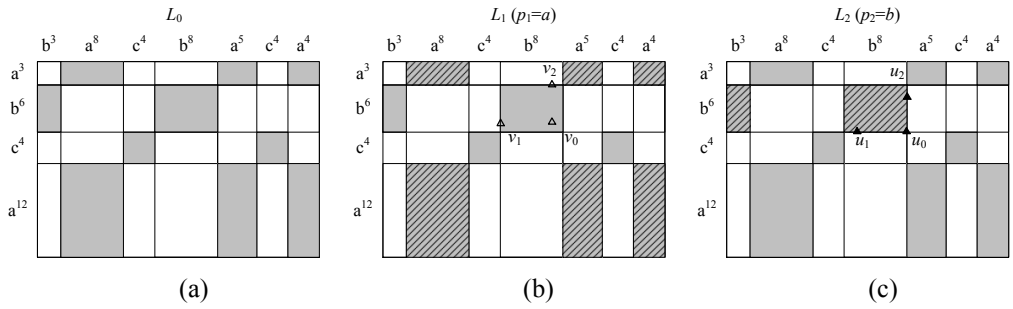

Fig. 1

AN EXAMPLE OF THE DP LATTICE SOLVING THE CLCS PROBLEM. (A) THE LOWEST LAYER, L_0. (B) L_1, WHERE $p_1 = a$. (C) L_2, WHERE $p_2 = b$.

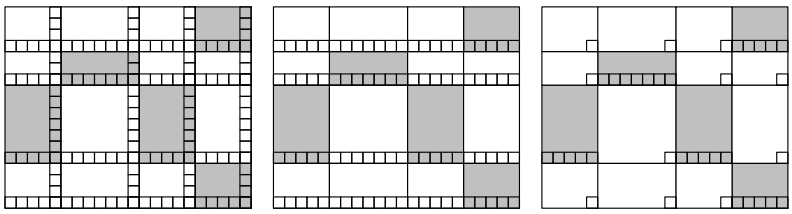

Fig. 2

(SEE [7]). THE ALGORITHMS WHICH SOLVE THE LCS PROBLEM FOR RLE STRINGS BY USING THE DP TECHNIQUE. (A) BUNKE AND CSIRIK [8]. (B) LIU ET AL. [10]. (C) ANN ET AL. [7].

which are encoded in RLE format into N, M and R runs, respectively, there exists an algorithm for computing the constrained LCS of X and Y with respect to P in $O(NMr+r\times\min\{q_1,q_2\}+q_3)$ time, where q_1 and q_2 denote the numbers of elements in the bottom and right boundaries of the partially matched blocks on the first layer L_0, and q_3 denotes the number of elements of whole boundaries of all fully matched cuboids in the DP lattice.

4. Conclusion

In this paper, we consider how to solve the CLCS problem more efficiently if the input strings and the constrained sequence can be compressed well in RLE format. For this purpose, we propose two algorithms which adopt the subset of the elements in the original DP lattice. We first propose a simple but slow algorithm, which solves the problem in $O(nmR+nMr+Nmr)$ time. Then we propose an improved algorithm by applying the concept of our previous work, which solves the problem in $O(NMr+r\times\min\{q_1,q_2\}+q_3)$ time, where q_1 and q_2 denote the numbers of elements in the bottom and right boundaries of the partially matched blocks on the first layer, and q_3 denotes the number of elements of whole boundaries of all fully matched cuboids in the DP lattice. If the compression ratio is good, our work obviously outperforms the previously known DP algorithm and the Hunt-and-Szymanski-like algorithm.

Acknowledgement

This research work was partially supported by the National Science Council of Taiwan under contract NSC-97-2221-E-110-064.

References

[1] R. Wagner and M. Fischer, "The string-to-string correction problem," *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, 1974.

[2] D. S. Hirschberg, "Algorithms for the longest common subsequence problem," *Journal of the ACM*, vol. 24, no. 4, pp. 664–675, 1977.

[3] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Communications of the ACM*, vol. 20, no. 5, pp. 350–353, 1977.

[4] C. Rick, "Simple and fast linear space computation of longest common subsequences," *Information Processing Letters*, vol. 75, no. 6, pp. 275–281, 2000.

[5] C. B. Yang and R. C. T. Lee, "Systolic algorithm for the longest common subsequence problem," *Journal of the Chinese Institute of Engineers*, vol. 10, no. 6, pp. 691–699, 1987.

[6] K. Sayoood and E. F. Eds., *Introduction to Data Compression*, 2nd ed. Morgan Kaufmann, Los Altos, CA, 2000.

[7] H. Y. Ann, C. B. Yang, C. T. Tseng, and C. Y. Hor, "A fast and simple algorithm for computing the longest common subsequence of run-length encoded strings," *Information Processing Letters*, vol. 108, pp. 360–364, 2008.

[8] H. Bunke and J. Csirik, "An improved algorithm for computing the edit distance of run-length coded strings," *Information Processing Letters*, vol. 54, no. 2, pp. 93–96, 1995.

[9] A. Apostolico, G. M. Landau, and S. Skiena, "Matching for run-length encoded strings," *Journal of Complexity*, vol. 15, no. 1, pp. 4–16, 1999.

[10] J. J. Liu, Y. L. Wang, and R. C. T. Lee, "Finding a longest common subsequence between a run-length-encoded string and an uncompressed string," *Journal of Complexity*, vol. 24, no. 2, pp. 173–184, 2008.
[11] Y. T. Tsai, "The constrained longest common subsequence problem," *Information Processing Letters*, vol. 88, no. 4, pp. 173–176, 2003.
[12] A. N. Arslan and Ö. Eğecioğlu, "Algorithms for the constrained longest common subsequence problems," *International Journal of Foundations Computer Science*, vol. 16, no. 6, pp. 1099–1109, 2005.
[13] F. Y. L. Chin, A. D. Santis, A. L. Ferrara, N. L. Ho, and S. K. Kim, "A simple algorithm for the constrained sequence problems," *Information Processing Letters*, vol. 90, no. 4, pp. 175–179, 2004.
[14] C. S. Iliopoulos and M. S. Rahman, "New efficient algorithms for the LCS and constrained LCS problems," *Information Processing Letters*, vol. 106, no. 1, pp. 13–18, 2008.
[15] P. van Emde Boas, "Preserving order in a forest in less than logarithmic time and linear space," *Information Processing Letters*, vol. 6, no. 3, pp. 80–82, 1977.
[16] V. Freschi and A. Bogliolo, "Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism," *Information Processing Letters*, vol. 90, pp. 167–173, 2004.