The Longest Common Subsequence Problem with the Gapped Constraint*

Kai-Yuan Cheng[a], Kuo-Si Huang[b] Chang-Biau Yang[a†]and Hsin-Yen Ann[c]

[a]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan

[b]Department of Information Management
National Kaohsiung Marine University, Kaohsiung, Taiwan

[c]National Center for High-Performance Computing, Tainan, Taiwan

Abstract

This paper considers a variant of the longest common subsequence (LCS) problem with the gapped constraint, called the LCSGC problem. Given two sequences A, B, and a constrained sequence C with a gap list, whose lengths are m, n, and r, respectively, the LCSGC problem is to find an LCS Z of A and B such that C is contained as a subsequence of Z and the gapped constraints corresponding to C are satisfied. In this paper, two algorithms with time complexities $O(m^2n^2r)$ and $O(mnr \times \min\{m, n\})$ are proposed based on the dynamic programming technique for solving the LCSGC problem.

Keywords: longest common subsequence, gapped constraint, dynamic programming, dominant list

1 Introduction

The *longest common subsequence* (LCS) problem has been widely studied for several decades and it has widely practical applications in computer science and computational biology [2, 7–10, 13, 14]. A sequence S' is said to be a *subsequence* of another sequence S if and only if S can be transformed into S' by deleting zero or more characters. Given two sequences A and B, sequence Z is said to be a *common subsequence* of A and B if Z is a subsequence of both A and B. Among the common subsequences of A and B, the LCS is the one with the longest length.

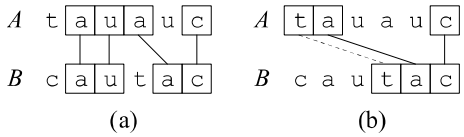

Figure 1: Examples of LCS and CLCS of $A =$ `tauauc`, $B =$ `cautac`, and a constrained sequence $C =$ `t`. (a)The LCS of A and B is `auac`. (b)The CLCS of A and B with the constrained sequence C is `tac`.

Several variants of the LCS problem have been proposed. In 2003, Tsai [15] first addressed the *constrained longest common subsequence* (CLCS) problem. Given two sequences A and B, and a constrained sequence C, the CLCS problem is to find an LCS Z of A and B such that C is included in Z as its subsequence. Figure 1 illustrates examples of LCS and CLCS. Later on, some researchers solved the CLCS problem based on the *dynamic programming* (DP) technique with a more efficient way [4, 6]. Both time and space complexities are $O(mnr)$, where $|A| = m$, $|B| = n$ and $|C| = r$.

Recently, Chen and Chao [5] generalized the CLCS problem to four problems, which are to find the LCS of A and B including/excluding C as a subsequence/substring, respectively. The four problems are SEQ-IC-LCS (including as a subsequence), SEQ-EC-LCS (excluding as a subsequence), STR-IC-LCS (including as a substring)

*This research was partially supported by the National Science Council of Taiwan under contract NSC 101-2221-E-022-018.

†To whom all correspondence should be sent: cbyang@cse.nsysu.edu.tw (Chang-Biau Yang).

and STR-EC-LCS (excluding as a substring), where SEQ-IC-LCS is the original CLCS problem. For SEQ-IC-LCS, Ann *et al.* [3] considered the three input sequences which are in run-length encoding format, a linear-time compressing technique, and greatly reduced the number of DP cells required to be evaluated. For STR-IC-LCS, Tseng *et al.* [16] proposed a more efficient algorithm for solving it with quadratic time. For STR-EC-LCS, Ann *et al.* [1] also proposed a more efficient algorithm for solving it with $O(mnr)$ time.

The *fixed gap longest common subsequence* (FGLCS) problem is another variant of the LCS problem [11, 12]. Given two sequences A and B, and the length of fixed gap g, the FGLCS problem is to find an LCS Z of A and B such that for any two consecutive characters z_i and z_{i+1} in Z, their corresponding distance in both A and B is at most $g+1$. The best-known algorithm for solving the FGLCS problem was proposed by Iliopoulos *et al.* [11], which requires $O(mn)$ time. In 2011, Peng and Yang [12] proposed a more flexible variant of the FGLCS problem, called the *variable gapped longest common subsequence* (VGLCS) problem, where the gapped constraint at each character of A or B is not fixed.

This paper considers the LCS problem of combined constraints, called the *longest common subsequence problem with the gapped constraint* (LCSGC), which combines the concepts of the CLCS and VGLCS problems. In the LCSGC problem, the inputs are two sequences A and B, and a constrained sequence C with a gapped constraint G_k applied to each c_k. The goal is to find an LCS Z of A and B such that C is contained as a subsequence of Z, and for any two consecutive constrained characters $c_k = z_p$ and $c_{k+1} = z_q$, $q - p \leq G_k + 1$. In other words, the gap between two consecutive characters c_k and c_{k+1} in the answer sequence is at most G_k.

This paper is organized as follows. Section 2 formally defines the LCSGC problem. We first propose a straightforward algorithm for solving the LCSGC problem in Section 3. Then, an improved algorithm is presented in Section 4. Finally, the conclusion of this paper is given in Section 5.

2 The Problem Definition

The *longest common subsequence problem with the gapped constraint* (LCSGC) involves the character constraint from the CLCS problem [15] and the gapped constraint from the VGLCS problem [12]. For a given string S, let s_i denote its ith character and $S_{i..j}$ denote the substring starting at s_i and ending at s_j. Given two sequences A and B, and a constrained sequence C with a gapped constraint G_k applied to c_k, $0 \leq k \leq r = |C|$, the LCSGC problem is that of finding an LCS Z of A and B such that C is included in Z as its subsequence, and for two consecutive constrained characters $c_k = z_p$ and $c_{k+1} = z_q$ in C, $q - p \leq G_k + 1$, where $1 \leq k \leq r-1$. Note that the lengths in the answer LCSGC before c_1 and after c_r are at most G_0 and G_r, respectively.

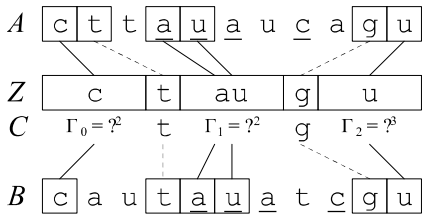

Figure 2: An example of LCSGC of A = `cttauaucagu`, B = `cautauatcgu`, and constrained sequence C = `tg` with gapped constraint $G = \{2, 2, 3\}$. The question mark ("?") represents a wildcard character for a matched pair in a gapped area. The answer is Z = `ctaugu` with length 6.

Figure 2 shows an example of the LCSGC problem, where A = `cttauaucagu`, B = `cautauatcgu`, and the constrained sequence C = `tg` with gapped constraint $G = \{2, 2, 3\}$. Note that $G_0 = 2$, $G_1 = 2$ and $G_2 = 3$ here. In Figure 2, the original CLCS between constrained character c_1=`t` and c_2=`g` is `auac`. However, with the gapped constraint, the *gapped area* Γ_1 between c_1 and c_2 can contain at most two characters since $G_1 = 2$. Thus, only two matched pairs between c_1 and c_2 are preserved.

3 A Straightforward Algorithm

Our straightforward algorithm for solving the LCSGC problem is based on the concept of the CLCS algorithm proposed by Tsai [15]. Let $H(x_1, x_2, y_1, y_2)$ denote the LCS length of substrings $A_{x_1..x_2}$ and $B_{y_1..y_2}$ of A and B, respectively. Let $M(i, j, k)$ denote the length of the LCSGC of $A_{1..i}$ and $B_{1..j}$ with gapped constraint $C_{1..k}$. The DP formula for computing $M(i, j, k)$ is given in Formula 1.

In Formula 1, $M(i, j, k)$ is positive only if $a_i = b_j = c_k$, and it means that there exists a sequence

$$M(i,j,k) = \begin{cases} 0 & \text{if } k = 0; \\ \min\{H(1,i-1,1,j-1), G_0\} + 1 & \text{if } k = 1 \text{ and } a_i = b_j = c_k; \\ \max_{1\le x\le i-1, 1\le y\le j-1}\{M(x,y,k-1) + \min\{H(x+1,i-1,y+1,j-1), G_{k-1}\} + 1\} & \text{if } a_i = b_j = c_k; \\ -\infty & \textit{otherwise.} \end{cases} \quad (1)$$

of LCSGC which satisfies the first k gapped constraint and the matched pair (a_i, b_j) is identical to c_k. For computing $M(i,j,k)$, Formula 1 searches all possible matches in the $(k-1)$th layer to form an LCS with constraints, and then selects the best one as the result.

However, Formula 1 only finds out all LCSGC sequences satisfying the gapped constraint whose last matched pair is identical to c_r. In fact, the last matched pair is not necessarily identical to c_r due to the problem definition. Therefore, Formula 2 is given for finding the final answer. Formula 2 computes the length of $M(i,j,r)$ concatenating with the LCS of $A_{i+1..m}$ and $B_{j+1..n}$ for each possible solution stored in the rth layer.

$$L_{ans} = \max_{1\le x\le m, 1\le y\le n}\{M(x,y,r) + \min\{G_r, H(x+1,m,y+1,n)\}\}. \quad (2)$$

The time complexity of Formula 1 depends on the computation of $M(i,j,k)$. Computation of each $H(x_1,x_2,y_1,y_2)$, $1 \le x_1 \le x_2 \le m$ and $1 \le y_1 \le y_2 \le n$, requires $O(1)$ query time if the $O(mn)$ dynamic programming (DP) lattices are constructed for all possible starting positions of A and B in the preprocessing phase, which needs $O(m^2n^2)$ time. For the kth layer, computing each $M(i,j,k)$ takes $O(mn)$ time because the number of matched pairs satisfying condition in the $(k-1)$th layer is at most $(i-1)\times(j-1)$, where $1 \le i \le m, 1 \le j \le n$. Thus, the time required for each layer is $O(m^2n^2)$ since each layer has $O(mn)$ lattice cells. As a summary, the time complexity of Formula 1 for solving the LCSGC problem is $O(m^2n^2r)$, obtained by summing the time required for all layers. Obviously, the time complexity of Formula 2 is $O(mn)$. In summary, the time complexity of this straightforward algorithm is $O(m^2n^2r + mn) = O(m^2n^2r)$.

4 An Improved Algorithm

It is not easy to reduce the time complexity of the straightforward algorithm. To design an improved algorithm, it requires a different perspective so that the time complexity could be reduced. The definition of *pair* and two data structures used in the improved algorithm are introduced in the following.

Definition 1. *A* pair (a_i, b_j) *is called a* match *if* $a_i = b_j$, *or a* mismatch *if* $a_i \neq b_j$. *Furthermore, it is called a* constrained match *if* $a_i = b_j = c_k$ *for some* k, *or a* normal match *if it is located in a gapped area.*

In the straightforward algorithm, $M(i,j,k)$ contains only one integer value for representing the LCSGC length of $A_{1..i}$ and $B_{1..j}$ with $C_{1..k}$, where (a_i, b_j) is a constrained match corresponding to c_k. In order to reduce the computation time, we have to enhance the content of each DP lattice cell. Now, each cell of the DP lattice stores a list of *dominant elements*. A dominant element records the current LCSGC length and its maximal length limit. The *maximal length limit* is the length of a common subsequence of $A_{1..i}$ and $B_{1..j}$ with gapped constraint $C_{1..k}$ plus the most possibly additional LCS length by considering the gapped constraint G_k after c_k in advance. It represents the length upper bound of a temporary LCSGC sequence under the gapped constraint. For example, suppose there is a temporary sequence of LCSGC with length t that satisfies the first k gapped constraints. It implies that c_k is the last character of this LCSGC and the additional possible LCS length of gapped area Γ_k is limited to G_k. Hence the maximal length limit of this LCSGC in the kth layer is initially to be set to $t + G_k$.

Definition 2. *An* element $e = (\lambda, \mu)$ *in lattice cell* $D(i,j,k)$ *represents the length and its maximal length limit of a common subsequence of* $A_{1..i}$ *and* $B_{1..j}$ *with gapped constraint* $C_{1..k}$, *where* $1 \le i \le m$, $1 \le j \le n$, $1 \le k \le r$.

According to Definition 2, $D(i,j,k)$ may contain more than one feasible solution with different lengths and maximal length limits. If one element is dominated by another, then the former will removed from the dominant list. That is, $D(i,j,k)$

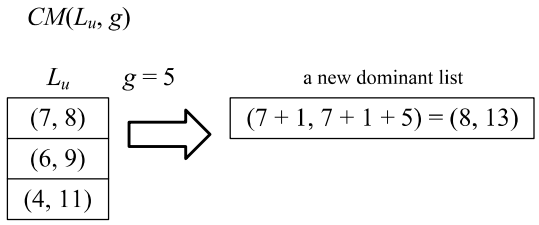

Figure 3: An example of function $CM(L_u, g)$ with $g = 5$.

contains a dominant list, formed by dominant elements.

Definition 3. *Suppose two elements* $e_x = (\lambda_x, \mu_x)$ *and* $e_y = (\lambda_y, \mu_y)$ *are both corresponding to* $A_{1..i}$, $B_{1..j}$ *and* $C_{1..k}$. *It is said that* e_x *dominates* e_y *either if* $\lambda_x > \lambda_y$ *and* $\mu_x \geq \mu_y$, *or if* $\lambda_x \geq \lambda_y$ *and* $\mu_x > \mu_y$.

For two elements $e_x = (\lambda_x, \mu_x)$ and $e_y = (\lambda_y, \mu_y)$ corresponding to $A_{1..i_1}$, $B_{1..j_1}$ and $C_{1..k}$, suppose that e_x dominates e_y. The LCSGC length expansion of e_x is better than e_y because for a path that contains t matches corresponding to $A_{i_1+1..i_2}$ and $B_{j_1+1..j_2}$, elements e_x and e_y will be expanded to $e'_x = (\lambda_x + t, \mu_x)$ and $e'_y = (\lambda_y + t, \mu_y)$ if their maximal length limits are not reached. Thus, if two elements e_x and e_y are in $D(i, j, k)$ and e_x dominates e_y, keeping e_x is enough for maintaining the information of LCSGC in this lattice cell.

Definition 4. *A* dominant list *consists of* dominant elements, *which are not dominated by any other element.*

We define three functions for maintaining the dominant list in each cell as follows.

Definition 5. *Let* λ_{Max} *denote the maximal LCSGC length of all elements in dominant list* L_u. *For a constrained match, the function* $CM(L_u, g)$ *is to find the element with the maximal length in* L_u, *and then creates a new list containing a single element* $(\lambda_{Max}+1, \lambda_{Max}+1+g)$. *In other words,* $CM(L_u, g) = \{(\lambda_{Max} + 1, \lambda_{Max} + 1 + g)\}$.

Figure 3 shows an example of function $CM(\ )$. Actually, $CM(L_u, g)$ is used to create a list for a constrained match. Let $L_{i,j,k}$ denote the dominant list in $D(i, j, k)$. Suppose (a_i, b_j) is a constrained match corresponding to c_k in $D(i, j, k)$. Then, we have

$$L_{i,j,k} = CM(L_{i-1,j-1,k-1}, G_k). \quad (3)$$

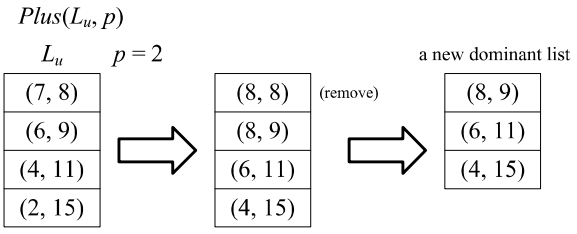

Figure 4: An example of function $Plus(L_u, p)$ with $p = 2$.

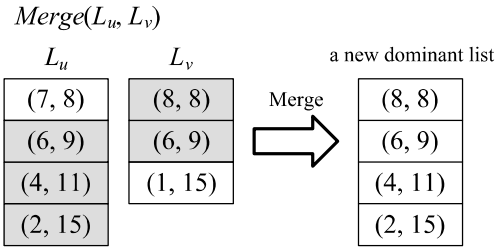

Figure 5: An example of function $Merge(L_u, L_v)$.

Definition 6. *The* Add *function applied to a dominant element* $e_x = (\lambda_x, \mu_x)$ *is defined as* $Add(e_x, p) = (\min\{\lambda_x + p, \mu_x\}, \mu_x)$. *The* Plus *function applied to a dominant list* L_u *is to apply Add function to each element, then to preserve all dominant elements.*

Figure 4 shows an example of function $Plus(\ )$. For a matched pair (a_i, b_j) in the kth layer, the pair can expand the length of each element of $L_{i-1,j-1,k}$ as a normal match. It can be accomplished by $Plus(L_{i-1,j-1,k}, 1)$.

Definition 7. *The function* $Merge(L_u, L_v)$ *first combines* L_u *and* L_v, *then preserves all dominant elements to form a new dominant list.*

Figure 5 shows an example of function $Merge(\ )$.

Based on the above three functions for maintaining dominant lists, our improved algorithm for solving LCSGC problem is given in Formula 4. For example, suppose $A =$ `cttauaucagu`, $B =$ `cautauatcgu` and $C =$ `tg` and gapped constraint $G = \{2, 2, 3\}$. The illustration of each layer for the improved algorithm is shown in Figure 6.

The time complexities of functions $CM(\ )$, $Plus(\ )$ and $Merge(\ )$ are all linear to the list length involved in the computation if a greedy method is applied to sorted lists. Furthermore, the size of each dominant list is bounded by $\min\{m, n\}$. Thus the overall time complexity of the improved algorithm is $O(mnr \times \min\{m, n\})$.

$$L_{i,j,k} = \begin{cases} \{(0, G_0)\} & \text{if } k = 0 \text{ and } (i = 0 \text{ or } j = 0); \\ Merge(CM(L_{i-1,j-1,k-1}, G_k), & \\ \qquad Plus(L_{i-1,j-1,k}, 1)) & \text{if } a_i = b_j = c_k; \\ Plus(L_{i-1,j-1,k}, 1) & \text{if } a_i = b_j \text{ and } a_i \neq c_k; \\ Merge(L_{i-1,j,k}, L_{i,j-1,k}) & \text{if } a_i \neq b_j. \end{cases} \tag{4}$$

	-	c	a	u	t	a	u	a	t	c	g	u
-	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)
c	(0, 2)	(1, 2)	(1, 2)	(1, 2)	(1, 2)	(1, 2)	(1, 2)	(1, 2)	(1, 2)	(1, 2)	(1, 2)	(1, 2)
t	(0, 2)	(1, 2)	(1, 2)	(1, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)
t	(0, 2)	(1, 2)	(1, 2)	(1, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)
a	(0, 2)	(1, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)
u	(0, 2)	(1, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)
a	(0, 2)	(1, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)
u	(0, 2)	(1, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)
c	(0, 2)	(1, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)
a	(0, 2)	(1, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)
g	(0, 2)	(1, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)
u	(0, 2)	(1, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	(2, 2)

Layer 0

	-	c	a	u	t	a	u	a	t	c	g	u
-												
c												
t					(2, 4)	(2, 4)	(2, 4)	(2, 4)	(2, 4)	(2, 4)	(2, 4)	(2, 4)
t					(2, 4)	(2, 4)	(2, 4)	(2, 4)	(3, 5)	(3, 5)	(3, 5)	(3, 5)
a					(2, 4)	(3, 4)	(3, 4)	(3, 4)	(3, 5)	(3, 5)	(3, 5)	(3, 5)
u					(2, 4)	(3, 4)	(4, 4)	(4, 4)	(4, 4) (3, 5)	(4, 4) (3, 5)	(4, 4) (3, 5)	(4, 4) (3, 5)
a					(2, 4)	(3, 4)	(4, 4)	(4, 4)	(4, 4) (3, 5)	(4, 4) (3, 5)	(4, 4) (3, 5)	(4, 4) (3, 5)
u					(2, 4)	(3, 4)	(4, 4)	(4, 4)	(4, 4) (3, 5)	(4, 4) (3, 5)	(4, 4) (3, 5)	(4, 4) (3, 5)
c					(2, 4)	(3, 4)	(4, 4)	(4, 4)	(4, 4) (3, 5)	(4, 5)	(4, 5)	(4, 5)
a					(2, 4)	(3, 4)	(4, 4)	(4, 4)	(4, 4) (3, 5)	(4, 5)	(4, 5)	(4, 5)
g					(2, 4)	(3, 4)	(4, 4)	(4, 4)	(4, 4) (3, 5)	(4, 5)	(5, 5)	(5, 5)
u					(2, 4)	(3, 4)	(4, 4)	(4, 4)	(4, 4) (3, 5)	(4, 5)	(5, 5)	(5, 5)

Layer 1

	-	c	a	u	t	a	u	a	t	c	g	u
-												
c												
t												
t												
a												
u												
a												
u												
c												
a												
g											(5, 8)	(5, 8)
u											(5, 8)	(6, 8)

Layer 2

Figure 6: An example of the improved algorithm for finding the LCSGC of A = `cttauaucagu`, B = `cautauatcgu` and C = `tg` with gapped constraint $G = \{2, 2, 3\}$.

5 Conclusion

This paper considers a variant of the classical LCS problem, called the longest common subsequence with the gapped constraint (LCSGC) problem. Given input sequences A, B, and constrained sequence C, whose lengths are m, n and r respectively, two algorithms with time complexities $O(m^2n^2r)$ and $O(mnr \times \min\{m, n\})$ are proposed based on the dynamic programming technique for solving the LCSGC problem. In the future, it is worthy to pay more attention on the details of the LCSGC problem such as the analysis of the average case and the lower bound.

References

[1] H.-Y. Ann, C.-B. Yang, and C.-T. Tseng, "Efficient polynomial-time algorithms for the constrained LCS problem with strings exclusion," *Journal of Combinatorial Optimization*, to appear.

[2] 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, no. 6, pp. 360–364, 2008.

[3] ——, "Fast algorithms for computing the constrained LCS of run-length encoded strings," *Theoretical Computer Science*, vol. 432, pp. 1–9, 2012.

[4] A. N. Arslan and Ö. Eğecioğlu, "Algorithms for the constrained longest common subsequence problems," *International Journal of Foundations of Computer Science*, vol. 16, no. 6, pp. 1099–1109, 2005.

[5] Y.-C. Chen and K.-M. Chao, "On the generalized constrained longest common subsequence problems," *Journal of Combinatorial Optimization*, vol. 21, pp. 383–392, 2011.

[6] 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.

[7] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Communications of the ACM*, vol. 18, no. 6, pp. 341–343, 1975.

[8] K.-S. Huang, C.-B. Yang, K.-T. Tseng, H.-Y. Ann, and Y.-H. Peng, "Efficient algorithms for finding interleaving relationship between sequences," *Information Processing Letters*, vol. 105, no. 5, pp. 188–193, 2008.

[9] K.-S. Huang, C.-B. Yang, K.-T. Tseng, Y.-H. Peng, and H.-Y. Ann, "Dynamic programming algorithms for the mosaic longest common subsequence problem," *Information Processing Letters*, vol. 102, no. 2-3, pp. 99–103, 2007.

[10] 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.

[11] C. S. Iliopoulos, M. Kubica, M. S. Rahman, and T. Waleń, "Algorithms for computing the longest parameterized common subsequence," in *Combinatorial Pattern Matching*, ser. Lecture Notes in Computer Science, B. Ma and K. Zhang, Eds. Springer Berlin Heidelberg, 2007, vol. 4580, pp. 265–273.

[12] Y.-H. Peng and C.-B. Yang, "The longest common subsequence problem with variable gapped constraints," in *Proceedings of the 28th Workshop on Combinatorial Mathematics and Computation Theory*, Penghu, Taiwan, 2011, pp. 17–23.

[13] Y.-H. Peng, C.-B. Yang, K.-S. Huang, C.-T. Tseng, and C.-Y. Hor, "Efficient sparse dynamic programming for the merged LCS problem with block constraints," *International Journal of Innovative Computing, Information and Control*, vol. 6, no. 4, pp. 1935–1947, 2010.

[14] Y.-H. Peng, C.-B. Yang, K.-S. Huang, and K.-T. Tseng, "An algorithm and applications to sequence alignment with weighted constraints," *International Journal of Foundations of Computer Science*, vol. 21, no. 1, pp. 51–59, 2010.

[15] Y.-T. Tsai, "The constrained longest common subsequence problem," *Information Processing Letters*, vol. 88, no. 4, pp. 173–176, 2003.

[16] C.-T. Tseng, C.-B. Yang, and H.-Y. Ann, "Efficient algorithms for the longest common subsequence problem with sequential substring constraints," *Journal of Complexity*, vol. 29, no. 1, pp. 44 – 52, 2013.