Vol.2019-AL-174 No.5
2019/9/17

高速な STR-EC-LCS アルゴリズム

山田 航平[1]　中島 祐人[2]　稲永 俊介[2]　坂内 英夫[2]　竹田 正幸[2]

概要：2 つの文字列 A, B の共通部分列のうち最長の文字列を，A, B の最長共通部分列（Longest Common Subsequence : LCS）といい，これを求める LCS 問題の類似問題群として，文字列 A と B の共通部分列でかつ制約文字列 P を 部分列 / 部分文字列 として 含む / 含まない ような最長の文字列を求める，GC-LCS 問題群がある．本研究では，これらのうち A と B の共通部分列でかつ P を部分文字列として含まない最長の文字列のひとつを求める STR-EC-LCS 問題を，$O(n|\Sigma| + (L'+1)(m-L'+1)r)$ 時間・領域で解くアルゴリズムを提案した．ただし，$A, B, P \in \Sigma^*$，$|A| = m$，$|B| = n$，$|P| = r$，L' は求める STR-EC-LCS の長さとし，一般性を失うことなく $m \leq n$ とする．$|\Sigma| \leq mr$ が成り立つ場合，これは既存の $O(mnr)$ 時間・領域で解く Wang らのアルゴリズム（Inf. Process. Lett. 2013）と比べ最悪でも同程度の時間で動作し，特に $L' = O(1)$ または $m - L' = O(1)$ のとき，計算時間は $O(n|\Sigma| + mr)$ となる．

キーワード：文字列アルゴリズム，制約付き最長共通部分列，動的計画法

1. 序論

1.1 LCS

文字列 A と B の共通部分列のうち最長の文字列を，A と B の最長共通部分列（Longest Common Subsequence，以降頭文字をとって LCS と呼ぶ）といい，類似文書の検索，文章校正，生命情報科学分野における DNA の塩基配列の類似度測定など，幅広い分野で応用されている．文字列 A と B の LCS のひとつを求める LCS 問題を解くアルゴリズムとしては，$O(mn)$ 時間・領域で解く動的計画法（Dynamic Programming，以降 DP と呼ぶ）を用いたアルゴリズムがよく知られている．ただし，$m = |A|$，$n = |B|$ である．一方，1992 年，Nakatsu らによって，文字列 A と B の LCS 問題を $O(n(m-L))$ 時間で解くアルゴリズムが提案された [1]．ただし，$m \leq n$ とし，L は求める LCS の長さを表す．これは先の $O(mn)$ 時間で動作する DP アルゴリズムに比べて，L の値が大きいとき高速に動作する．これらの他にも，LCS については盛んに研究されてきた [2], [3], [4], [5], [6].

1.2 GC-LCS

LCS が DNA の塩基配列の類似度測定に応用されているのは 1.1 節で述べた通りであるが，特に分子生物学分野において，異なる種同士の DNA の塩基配列中に特定のパターンが出現する場合があり，これを類似度測定に取り入れたいという要求がある．そこで近年では，LCS 問題の派生として，文字列 A, B, P に対して，A と B の共通部分列であって P を 部分文字列 / 部分列 として 含む / 含まない 最長の文字列を求めるという 4 種の問題からなる問題群，Generalized Constrained - LCS（GC-LCS）問題群が考えられており [7]，それぞれ STR-IC-LCS，STR-EC-LCS，SEQ-IC-LCS，SEQ-EC-LCS 問題と呼ぶ．それぞれの現在知られている最も効率の良い DP アルゴリズムの時間計算量を**表 1** に記す．ただし，表 1 中において，$m = |A|$，$n = |B|$，$r = |P|$，m' と n' はそれぞれ A と B の連長圧縮長を表す．

表 1　現在知られている GC-LCS 問題群に対する最も効率の良い DP アルゴリズムの時間計算量．

問題	時間計算量
STR-IC-LCS	$O(m'n + mn')$ [8]
STR-EC-LCS	$O(mnr)$ [9]
SEQ-IC-LCS	$O(m + n + r\min\{m'n, mn'\})$ [10]
SEQ-EC-LCS	$O(mnr)$ [11]

特に，GC-LCS 問題群のうち有限アルファベット Σ 上の文字列 A, B, P に対して，A と B の共通部分列であって P を部分文字列として含まない最長の文字列のひとつを求める問題を，STR-EC-LCS 問題という．本研究では，STR-EC-LCS 問題に先述の Nakatsu らの LCS アルゴリズムを応用することで，$O(n|\Sigma| + (L'+1)(m-L'+1)r)$ 時

[1] 九州大学 大学院システム情報科学府
[2] 九州大学 大学院システム情報科学研究院

間・領域で解くアルゴリズムを提案する．ただし，$m \leq n$ とし，L' は求める STR-EC-LCS の長さを表す．$|\Sigma| \leq mr$ が成り立つ場合，これは既存の Wang らの STR-EC-LCS アルゴリズムと比べて最悪時でも同程度の計算時間で動作し，特に $L' = O(1)$ あるいは $m - L' = O(1)$ のときに効率的に動作する．そのときの時間計算量は $O(n|\Sigma| + mr)$ である．

2. 準備

本章では，本稿で用いる記号と用語の定義を述べる．

2.1 文字列

Σ を空でない有限アルファベットとし，Σ^* の要素を文字列と呼ぶ．文字列 A の長さを $|A|$ と記す．長さ 0 の文字列を空文字列と呼び，ε と記す．$1 \leq i \leq |A|$ を満たす任意の i について，$A[i]$ は A の i 番目の文字を表し，$1 \leq i \leq i' \leq |A|$ を満たす任意の i, i' について，$A[i..i']$ は A の $A[i]$ から $A[i']$ までの部分文字列を表す．特に，$1 \leq i \leq |A|$ を満たす任意の i について，$A[1..i]$ を A の接頭辞と呼び，$A[i..|A|]$ を A の接尾辞と呼ぶ．A から任意個の文字を取り去った文字列を A の部分列と呼ぶ．文字列 A の部分列であり，文字列 B の部分列でもあるような文字列を A と B の共通部分列と呼ぶ．

2.2 LCS 問題

文字列 A, B について，ある文字列 Z が A と B の共通部分列のうち最長の文字列であるとき，Z を A と B の LCS であるという．例えば，$A = \mathtt{ababc}, B = \mathtt{aacacb}$ の LCS は aab と aac である．また，LCS 問題を以下で定義する．

問題 1. 文字列 A, B が与えられたとき，A と B の LCS のひとつを求める．

2.3 STR-EC-LCS 問題

文字列 A, B と制約文字列 P について，ある文字列 Z が A と B の共通部分列で P を部分文字列として含まない最長の文字列であるとき，Z を A, B, P の STR-EC-LCS であるという．例えば，$A = \mathtt{ababc}, B = \mathtt{aacacb}, P = \mathtt{aa}$ の STR-EC-LCS は ab と ac である．このとき，A と B の LCS は aab と aac であるが，これらは $P = \mathtt{aa}$ を部分文字列として含んでいるため，STR-EC-LCS はこれらよりも長さが短くなる．また，STR-EC-LCS 問題を以下で定義する．

問題 2. 文字列 A, B と制約文字列 P が与えられたとき，A, B, P の STR-EC-LCS のひとつを求める．

3. 既存手法

本章では，本研究において参考にした既存研究について，簡単に紹介する．3.1 節では，Nakatsu らの提案した LCS 問題を解く DP アルゴリズム [1] に若干の変更を加えたものを紹介する．3.2 節では，Wang らの提案した STR-EC-LCS 問題を解く DP アルゴリズム [9] を紹介する．

$s \backslash i$	0	1	2	3	4	5	6	7	
0	0	0	0	0	0	0	0	0	
1	∞	2	2	1	1	1	1	1	
2	∞	∞	3	3	2	2	2	2	
3	∞	∞	∞	4	4	4	3	3	
4	∞	∞	∞	∞	7	6	6	4	
5	∞	∞	∞	∞	∞	∞	7	7	$L = 5$
6	∞	∞	∞	∞	∞	∞	∞	∞	
7	∞	∞	∞	∞	∞	∞	∞	∞	

図 1 $A = \mathtt{aabacab}, B = \mathtt{baabbcaa}$ のときの $d(i, s)$ の値．

3.1 Nakatsu らの LCS アルゴリズム

本節では，Nakatsu らの提案した LCS 問題を解く DP アルゴリズムに，4 章のわかりやすさの為に若干の変更を加えたものを紹介する．具体的にどのような変更を施したかについては，本節の最後に説明する．LCS 問題のそれぞれ長さ m, n の入力文字列 A, B について，一般性を失うことなく $m \leq n$ とする．$0 \leq i, s \leq m$ を満たす任意の i, s に対して，文字列 B 中の位置 $d(i, s)$ を，以下で定義する．

定義 1. $d(i, s)$ は，$A[1..i]$ と $B[1..j]$ の LCS の長さが s であるような，B 中で最小の位置 j である．ただし，任意の i について $d(i, 0) = 0$ とし，上記を満たす j が存在しない場合，$d(i, s) = \infty$ とする．

定義 1 から，任意の i, s について $i < s$ であれば $d(i, s) = \infty$ となる．例えば，$A = \mathtt{aabacab}, B = \mathtt{baabbcaa}$ のときの $d(i, s)$ の値は**図 1** のようになる．定義 1 から，求める LCS の長さを L とすると，L は $d(i, s) \neq \infty$ が存在する最大の s と等しい．したがって，任意の i, s について $d(i, s)$ を求めることができれば，L の値がわかる．図 1 の例では，$d(i, s) \neq \infty$ が存在する最大の s は 5 であるので，$L = 5$ である．

また，$d(i, s)$ について，

$$d(i, s) = \begin{cases} \min\{j, d(i-1, s)\} & \text{(1a)} \\ d(i-1, s) & \text{(1b)} \end{cases}$$

が成り立つ．ただし場合分けの条件は，

1a: $A[i] = B[j]$ かつ $j > d(i-1, s-1)$ を満たす B 中で最小の位置 $j \neq \infty$ が存在するとき

1b: それ以外

である．Nakatsu らのアルゴリズムでは，式 1a，1b を用いて $d(i-1, s-1)$ と $d(i-1, s)$ から $d(i, s)$ を動的に求め

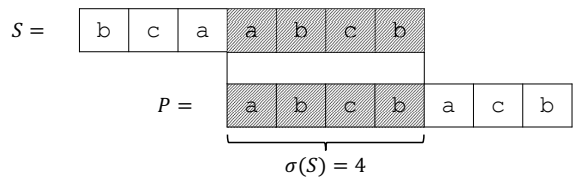

図 2　$P = \texttt{abcbacb}, S = \texttt{bcaabcb}$ のときの $\sigma(S)$.

ることで，A と B の LCS の長さを求める．なお，本稿で加えた若干の変更について，元の Nakatsu らのアルゴリズムでは，B 中の位置 $L_i(k)$ を DP を用いて求める．ただし，$L_i(k)$ の定義は以下の通りである．

定義 2. $L_i(k)$ は，$A[i..m]$ と $B[h..n]$ の LCS の長さが k であるような，B 中で最大の位置 h である．

3.2 Wang らの STR-EC-LCS アルゴリズム

本節では，Wang らの提案した STR-EC-LCS 問題を解く DP アルゴリズムを紹介する．まず，制約文字列 P に関して，関数 σ を以下で定義する．

定義 3. $\sigma(S)$ は，P の接頭辞でかつ S の接尾辞であるような最長の文字列の長さである．

例えば，$P = \texttt{abcbacb}, S = \texttt{bcaabcb}$ のとき，P の接頭辞でかつ S の接尾辞であるような最長の文字列は abcb であるので，$\sigma(S) = 4$ である（**図 2**）．

次に，STR-EC-LCS 問題のそれぞれ長さ m, n の入力文字列 A, B と，制約文字列 P に対して，それぞれ $1 \le i \le m, 1 \le j \le n, 0 \le k < r$ を満たす任意の i, j, k について $f(i, j, k)$ を以下で定義する．

定義 4. $f(i, j, k)$ は，以下を満たすような最長の文字列 Z の長さである．

- $A[1..i]$ と $B[1..j]$ の共通部分列である．
- P を部分文字列として含まない．
- $\sigma(Z) = k$

ただし，任意の i, j, k について，$f(i, 0, k) = f(0, j, k) = 0$.

$f(i, j, k)$ の定義から，f と求める STR-EC-LCS の長さ L' について，

$$L' = \max_{0 \le t < r} \{f(m, n, t)\}$$

が成り立つため，任意の i, j, k について $f(i, j, k)$ が求められれば，L' の値がわかる．また，$f(i, j, k)$ について，

$$f(i,j,k) = \begin{cases} \max\{f(i-1,j,k), f(i,j-1,k)\} & \text{(2a)} \\ \max\{f(i-1,j-1,k), & \\ \quad 1 + \max_{0 \le t < r} \{f(i-1,j-1,t) & \text{(2b)} \\ \quad \mid \sigma(P[1..t]A[i]) = k\}\} & \end{cases}$$

が成り立つ．ただし場合分けの条件は，

2a: $A[i] \neq B[j]$ のとき

2b: $A[i] = B[j]$ のとき

である．Wang らのアルゴリズムでは，式 2a , 2b を用いて $f(i-1, j, k), f(i, j-1, k), f(i-1, j-1, 0), \cdots, f(i-1, j-1, r-1)$ から $f(i, j, k)$ を動的に求めることで，A, B, P の STR-EC-LCS の長さを求める．

4. 提案手法

本章では，3 章で紹介したアルゴリズムに基づいた STR-EC-LCS 問題を解く DP アルゴリズムを提案する．4.1 節では，我々の提案する DP アルゴリズムの要となる漸化式を示す．4.2 節では，4.1 節で示した漸化式を元にしたアルゴリズムを示す．4.3 節では，4.2 節で示したアルゴリズムの計算量を示す．

4.1 漸化式

3.1 節と同様に，以降，それぞれ長さ m, n, r の STR-EC-LCS 問題の入力文字列 A, B と制約文字列 P について，$m \le n$ であるとする．また $m < r$ のとき，A, B, P の STR-EC-LCS は A, B の LCS と一致するため，以降 $r \le m \le n$ であるとする．$0 \le i \le m, 0 \le s \le m, 0 \le k < r$ を満たす任意の i, s, k について，まず今後の便利のために以下の条件を定義する．

定義 5. 文字列 Z が以下の 4 つの条件

- Z が $A[1..i]$ の部分列である
- Z が P を部分文字列として含まない
- $|Z| = s$
- $\sigma(Z) = k$

を満たすとき，Z は Property(i, s, k) を満たすという．

ただし，関数 σ は定義 3 で与えたものである．つぎに，文字列 B 中の位置 $d(i, s, k)$ を以下で定義する．

定義 6. $d(i, s, k)$ は Property(i, s, k) を満たし，かつ $B[1..j]$ の部分列であるような文字列 Z が存在する最小の位置 j である．ただし，そのような位置 j が存在しない場合，$d(i, s, k) = n + 1$ とする．

定義 6 から，任意の i と $k \neq 0$ を満たす任意の k について $d(i, 0, k) = n + 1$，$i < s$ を満たす任意の i, s, k について $d(i, s, k) = n + 1$ である．例として，$A = \texttt{aabacab}, B = \texttt{baabbcaa}, P = \texttt{aab}$ のときの $d(i, s, k)$ の値を**図 3** に示す．定義 6 から，求める STR-EC-LCS の長さを L' とすると，L' は $d(i, s, k) \neq n+1$ が存在する最大の s と等しい．したがって，任意の i, s, k について $d(i, s, k)$ が求まれば，L' の値がわかる．図 3 の例では，$d(i, s, k) \neq 9$ が存在する最大の s は 4 であるので，$L' = 4$ である．

以下に，$d(i, s, k)$ に関する漸化式を示す．

定理 1. $d(i, s, k)$ に関して，以下が成り立つ．

$$d(i, s, k) = \min(\{d(i-1, s, k)\} \cup \{j_t \mid 0 \le t < r\}) \quad (3)$$

ただし，j_t は以下を満たすような B 中で最小の位置 j である．

- $A[i] = B[j]$

$k=0$

s \ i	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	9	9	9	1	1	1	1	1
2	9	9	9	4	4	4	4	4
3	9	9	9	9	9	6	6	4
4	9	9	9	9	9	9	9	9
5	9	9	9	9	9	9	9	9
6	9	9	9	9	9	9	9	9
7	9	9	9	9	9	9	9	9

$k=1$

s \ i	0	1	2	3	4	5	6	7
0	9	9	9	9	9	9	9	9
1	9	2	2	2	2	2	2	2
2	9	9	9	9	2	2	2	2
3	9	9	9	9	7	7	7	7
4	9	9	9	9	9	9	7	7
5	9	9	9	9	9	9	9	9
6	9	9	9	9	9	9	9	9
7	9	9	9	9	9	9	9	9

$k=2$

s \ i	0	1	2	3	4	5	6	7
0	9	9	9	9	9	9	9	9
1	9	9	9	9	9	9	9	9
2	9	9	3	3	3	3	3	3
3	9	9	9	9	9	9	3	3
4	9	9	9	9	9	9	8	8
5	9	9	9	9	9	9	9	9
6	9	9	9	9	9	9	9	9
7	9	9	9	9	9	9	9	9

$L'=4$

図 3 $A=\texttt{aabacab}, B=\texttt{baabbcaa}, P=\texttt{aab}$ のときの $d(i,s,k)$ の値.

- $j > d(i-1, s-1, t)$
- Property$(i-1, s-1, t)$ を満たし，かつ $\sigma(ZA[i]) = k$ を満たす文字列 Z が存在する

そのような j が存在しない場合，$j_t = n+1$ とする．

証明.

$$d(i,s,k) \leq \min(\{d(i-1,s,k)\} \cup \{j_t \mid 0 \leq t < r\})$$

かつ

$$d(i,s,k) \geq \min(\{d(i-1,s,k)\} \cup \{j_t \mid 0 \leq t < r\})$$

を示す．

(i) $d(i,s,k) \leq \min(\{d(i-1,s,k)\} \cup \{j_t \mid 0 \leq t < r\})$

まず，定義 6 から，Property$(i-1,s,k)$ を満たし，$B[1..d(i-1,s,k)]$ の部分列であるような文字列 Z が存在する．この Z は $A[1..i]$ と $B[1..d(i-1,s,k)]$ の共通部分列でもあるため Property(i,s,k) も満たす．したがって，定義 6 から，

$$d(i,s,k) \leq d(i-1,s,k)$$

が成り立つ．次に，各 j_t について，j_t の定義から Property(i,s,k) を満たし，$B[1..j_t]$ の部分列であるような文字列 Z_t がそれぞれ存在する．したがって，定義 6 から，任意の t について

$$d(i,s,k) \leq j_t$$

が成り立つ．以上から

$$d(i,s,k) \leq \min(\{d(i-1,s,k)\} \cup \{j_t \mid 0 \leq t < r\})$$

が成り立つ．

(ii) $d(i,s,k) \geq \min(\{d(i-1,s,k)\} \cup \{j_t \mid 0 \leq t < r\})$

$$d(i,s,k) < \min(\{d(i-1,s,k)\} \cup \{j_t \mid 0 \leq t < r\})$$

と仮定し，背理法で示す．

$d(i,s,k) \neq n+1$ と仮定すると，定義 6 から，Property(i,s,k) を満たし，$B[1..d(i,s,k)]$ の部分列であるような文字列 Z が存在する．この Z から末尾の文字を取り去った文字列を Z' とすると，Z' は

- $A[1..i-1]$ の部分列である
- P を部分文字列として含まない
- $|Z'| = s-1$

を満たす．Z' について $\sigma(Z') = k'$ とすると，Z' は Property$(i-1,s-1,k')$ を満たすため，定義 6 から

$$d(i-1,s-1,k') < d(i,s,k)$$

が成り立つ．ここで Z' は Property$(i-1,s-1,k')$ を満たし，かつ $\sigma(Z'Z[d(i,s,k)]) = k$ を満たすため，$A[i] = B[d(i,s,k)]$ と仮定すると，$d(i,s,k)$ は $j_{k'}$ の 3 つの条件を満たす．$j_{k'}$ はそのような B 中の位置のうち最小のものであるため，

$$j_{k'} \leq d(i,s,k)$$

が成り立つ．ところが，これは最初に置いた仮定に矛盾しているため，$A[i] = B[d(i,s,k)]$ という先の仮定が誤っていることになり，

$$A[i] \neq B[d(i,s,k)]$$

が導かれる．したがって Z は $A[1..i]$ と $B[1..d(i,s,k)-1]$ の共通部分列であるか，$A[1..i-1]$ と $B[1..d(i,s,k)]$ の共通部分列であるかのいずれかである．

前者の場合 $d(i,s,k)$ の定義に矛盾し，後者の場合定義 6 から $d(i,s,k) = d(i-1,s,k)$ となり，仮定に矛盾する．

したがって仮定が誤っていたことになり，

$$d(i,s,k) \geq \min(\{d(i-1,s,k)\} \cup \{j_t \mid 0 \leq t < r\})$$

が導かれる．

(i)，(ii) から，定理 1 が成立する． □

4.2 アルゴリズム

定理 1 の漸化式を用いて STR-EC-LCS を計算するアル

アルゴリズム 1 提案アルゴリズム

Input: $A, B, P \in \Sigma^*$ $(|A| = m, |B| = n, m \leq n, |P| = r)$
Output: A, B, P の STR-EC-LCS のひとつ

1: 配列 next_B を構築
2: 配列 next_σ を構築
3: **for all** $0 \leq i \leq m$, $1 \leq k < r$ **do**
4: $d(i, 0, 0) \leftarrow 0$
5: $d(i, 0, k) \leftarrow n + 1$
6: **end for**
7: **for all** $1 \leq t \leq m, 0 \leq k < r$ **do**
8: $d(t - 1, t, k) \leftarrow n + 1$
9: **end for**
10: $S \leftarrow 0$
11: **for** $t = 1$ to $m - S + 1$ **do**
12: $i \leftarrow t$
13: $s \leftarrow 1$
14: **while** $s < m - t + 1$ かつ すべての k について $d(i - 1, s - 1, k) \neq n + 1$ **do**
15: **for** $k = 0$ to $r - 1$ **do**
16: $d(i, s, k) \leftarrow d(i - 1, s, k)$
17: $\delta_s(i, s, k) \leftarrow s$, $\delta_k(i, s, k) \leftarrow k$
18: **end for**
19: **for** $k = 0$ to $r - 1$ **do**
20: **if** $d(i - 1, s - 1, k) \neq n + 1$ **then**
21: $k' \leftarrow \mathrm{next}_\sigma(k, A[i])$
22: **if** $k' \neq r$ **then**
23: $j \leftarrow \mathrm{next}_B(d(i - 1, s - 1, k), A[i])$
24: **if** $j < d(i, s, k')$ **then**
25: $d(i, s, k') \leftarrow j$
26: $\delta_s(i, s, k') \leftarrow s - 1$, $\delta_k(i, s, k') \leftarrow k$
27: **if** $s > S$ **then**
28: $I \leftarrow i$, $S \leftarrow s$, $K \leftarrow k'$
29: **end if**
30: **end if**
31: **end if**
32: **end if**
33: **end for**
34: $i \leftarrow i + 1$
35: $s \leftarrow s + 1$
36: **end while**
37: **end for**
38: $L' \leftarrow S$
39: $i \leftarrow I$, $s \leftarrow S$, $k \leftarrow K$
40: **while** $s \neq 0$ **do**
41: $Z(s) \leftarrow B[d(i, s, k)]$
42: **while** $\delta_s(i, s, k) = s$ **do**
43: $i \leftarrow i - 1$, $s \leftarrow \delta_s(i, s, k)$, $k \leftarrow \delta_k(i, s, k)$
44: **end while**
45: $i \leftarrow i - 1$, $s \leftarrow \delta_s(i, s, k)$, $k \leftarrow \delta_k(i, s, k)$
46: **end while**
47: **return** $Z(1) \cdots Z(L')$

ゴリズムを，アルゴリズム 1 に示す．

アルゴリズム 1 では

(1) 各 $d(i, s, k)$ には，最初 $d(i - 1, s, k)$ の値を格納しておく．
(2) 各 $d(i - 1, s - 1, k)$ に対して，$d(i - 1, s - 1, k)$ 以降で初めて文字 $A[i]$ が現れる位置 j を求める．
(3) $j < d(i, s, \sigma(P[1..k]A[i]))$ のとき $d(i, s, \sigma(P[1..k]A[i]))$ を j に更新．

という操作をすべての k について行うことで，すべての k について $d(i, s, k)$ を求めるという方法をとっている．操作 2 において，配列 next_B を用いて計算時間の削減を図る．配列 next_B の定義は以下の通りである．

定義 7. $\mathrm{next}_B(j, \alpha) = \min_{j < j' \leq n}\{j' \mid B[j'] = \alpha\}$

すなわち，$\mathrm{next}_B(j, \alpha)$ は，B 中の位置 $j + 1$ 以降で初めて文字 α が出現するような位置である．ただし，位置 $j+1$ 以降に文字 α が出現しない場合，$\mathrm{next}_B(j, \alpha) = n + 1$ とする．なお，配列 next_B はアルゴリズム 2 によって構築する．

操作 2 において，

$$j = \mathrm{next}_B(d(i - 1, s - 1, k), A[i])$$

であるので，前もって next_B を計算しておけば，操作 2 を効率的に行うことができる．操作 3 においては，配列 next_σ を用いて計算時間の削減を図る．配列 next_σ の定義は以下の通りである．

定義 8. $next_\sigma(k, \alpha) = \sigma(P[1..k]\alpha)$

配列 next_σ はアルゴリズム 3 によって構築する．

なお，この配列 next_σ を用いて計算時間を削減するというアイデアと next_σ の構築法は 3 章で紹介した Wang らのアルゴリズムを参考にしている．操作 3 において，

$$\sigma(P[1..k]A[i]) = \mathrm{next}_\sigma(k, A[i])$$

であるため，任意の $\alpha \in \Sigma$ と任意の k について，配列 next_σ を前もって構築しておくことで，操作 3 における $\sigma(P[1..k]A[i])$ の値の計算を効率的に行うことができる．

また，アルゴリズム 1 における 11 行目の t に関してのループについて，t 番目のループでは $d(t, 1, 0)$ から $d(t, m-t, r-1)$ までの計算を行う．いま t 番目のループが終わった時点で

アルゴリズム 2 配列 next_B を構築するアルゴリズム

Input: B, Σ $(|B| = n)$
Output: next_B

1: **for all** $\alpha \in \Sigma$ **do**
2: $\mathrm{next}_B(n, \alpha) = n + 1$
3: **end for**
4: **for** $j = n - 1$ to 0 **do**
5: **for all** $\alpha \in \Sigma$ **do**
6: **if** $\alpha = B[j + 1]$ **then**
7: $\mathrm{next}_B(j, \alpha) = j + 1$
8: **else**
9: $\mathrm{next}_B(j, \alpha) = \mathrm{next}_B(j + 1, \alpha)$
10: **end if**
11: **end for**
12: **end for**
13: **return** next_B

アルゴリズム 3 配列 next_σ を構築するアルゴリズム

Input: $P, \Sigma \quad (|P| = r)$
Output: next_σ

```
 1: kmp(0) ← −1
 2: kmp(1) ← 0
 3: k ← 0
 4: for i = 2 to r do
 5:    while k ≥ 0 かつ P[k + 1] ≠ P[i] do
 6:       k ← kmp(k)
 7:    end while
 8:    k ← k + 1
 9:    kmp(i) ← k
10: end for
11: next_σ(0, P[1]) ← 1
12: for all α ∈ Σ かつ α ≠ P[1] do
13:    next_σ(0, α) ← 0
14: end for
15: for k = 1 to r − 1 do
16:    for all α ∈ Σ do
17:       if α = P[k + 1] then
18:          next_σ(k, α) ← k + 1
19:       else
20:          next_σ(k, α) ← next_σ(kmp(k), α)
21:       end if
22:    end for
23: end for
24: return next_σ
```

STR-EC-LCS の長さが少なくとも l であるとわかっていたとすると，$m-l+1$ 番目のループでは $d(m-l+1,1,0)$ から $d(m-l+1,l-1,r-1)$ までの計算しか行われないため，A, B, P の STR-EC-LCS のひとつを求めるという目的のもとでは，$m-L'+1$ 番目のループ以降は行う必要がない．ただし，L' は求める STR-EC-LCS の長さである．例として，$A = \mathtt{aabacab}, B = \mathtt{baabbcaa}, P = \mathtt{aab}$ に対してアルゴリズム 1 を動作させた場合の $t = 2$ のループが終了した時点での DP 表の様子を図 **4** に示す．この例では $L' \geq 3$ であることがわかっているので，図 4 の網かけ部の計算は行う必要がない．

アルゴリズム 1 では，以上のような計算する必要のない箇所の計算をしないことによって計算時間の削減を図っている．アルゴリズム 1 の 39 行目以降では，できた $d(i,s,k)$ の表から，長さ L' を達成する STR-EC-LCS のひとつを求めている．ただし，アルゴリズム 1 における配列 δ_s, δ_k は，各 $d(i,s,k)$ が $d(i-1,s,k)$ と $d(i-1,s-1,0), \cdots, d(i-1,s-1,r-1)$ のうちどれから得られたかを保持しておくための配列であり，各 $d(i,s,k)$ が更新されたときに同時に更新する．例として，$A = \mathtt{aabacab}, B = \mathtt{baabbcaa}, P = \mathtt{aab}$ に対する $d(i,s,k)$ の表と $\delta_s(i,s,k)$ と $\delta_k(i,s,k)$ から，STR-EC-LCS のひとつを求める様子を，図 **5** に示す．なお，図 5 中の矢印は $\delta_s(i,s,k), \delta_k(i,s,k)$ を表し，$d(i,s,k)$ から $d(i',s',k')$ への矢印は，$\delta_s(i,s,k) = s'$ かつ $\delta_k(i,s,k) = k'$ であることを表している．この例では，A, B, P の STR-EC-LCS のひとつ $Z = B[2]B[4]B[6]B[7] = \mathtt{abca}$ を求めている．

4.3 計算量

定理 2. アルゴリズム 1 は，文字列 A, B と制約文字列 P の STR-EC-LCS 問題を，$O(n|\Sigma| + (L'+1)(m-L'+1)r)$ 時間・領域で解く．ただし，$|A| = m, |B| = n, m \leq n, |P| = r$ であり，L' は求める STR-EC-LCS の長さである．

証明. 時間計算量: 配列 next_B はアルゴリズム 2 を用いることで $O(n|\Sigma|)$ 時間で構築でき，配列 next_σ はアルゴリズム 3 を用いることで $O(r|\Sigma|)$ 時間で構築できる [9]．アルゴリズム 111 行目の t に関してのループはちょうど $m-L'+1$ 回実行され，14 行目の while ループについては，ループを 1 度経るごとに s の値が 1 ずつ増えていくのと，定義 6 から任意の i, k と $s > L'$ を満たす s について $d(i,s,k) = n+1$ であるので，t に関してのループ 1 回に対して高々 $L'+1$ 回しか実行されない．15-35 行目の処理は $O(r)$ 時間で可能であるため，11 行目の t に関するループは $O(n|\Sigma| + (L'+1)(m-L'+1)r)$ 時間で実行できる．39 行目以降の処理については，40 行目の while ループを 1 回経るごとに i の値が 1 ずつ減っていくことから，$O(m)$ 時間かかる．したがって，全体の計算時間は $O(n|\Sigma| + (L'+1)(m-L'+1)r)$ である．
空間計算量: 配列 next_B は $O(n|\Sigma|)$ 領域，配列 next_σ は $O(r|\Sigma|)$ 領域，$d(i,s,k)$ の DP 表は，$\delta_s(i,s,k), \delta_k(i,s,k)$ も含めて $O((L'+1)(m-L'+1)r)$ 領域．したがって，全体で $O(n|\Sigma| + (L'+1)(m-L'+1)r)$ 領域である． □

時間計算量について，$L' = O(m)$ かつ $m - L' = O(m)$ であることから，提案手法の時間計算量は常に $O(n|\Sigma| + m^2 r)$ となるが，$|\Sigma| \leq mr$ という仮定のもとでは，これは Wang らの既存手法の時間計算量 $O(mnr)$ に比べて小さい．さらに $L' = O(1)$ または $m - L' = O(1)$ という条件下では，提案手法は Wang らの既存手法と比べより効率的に動作し，その場合の時間計算量は $O(n|\Sigma| + mr)$ となる．

5. 結論

本研究では，有限アルファベット Σ 上のそれぞれ長さ m, n, r の文字列 A, B, P の STR-EC-LCS 問題を，Nakatsu らの LCS 問題を解くアルゴリズムを応用することで，$O(n|\Sigma| + (L'+1)(m-L'+1)r)$ 時間・領域で解くアルゴリズムを提案した．ただし，L' は求める STR-EC-LCS の長さである．$|\Sigma| \leq mr$ が成り立つ場合，提案手法は既存の Wang らの STR-EC-LCS アルゴリズムと比べて，最悪時でも同程度の計算時間で動作し，特に $L' = O(1)$ または $m - L' = O(1)$ のときに効率的に動作する．今後の課題としては，STR-EC-LCS 問題をより高速に解くアルゴリズ

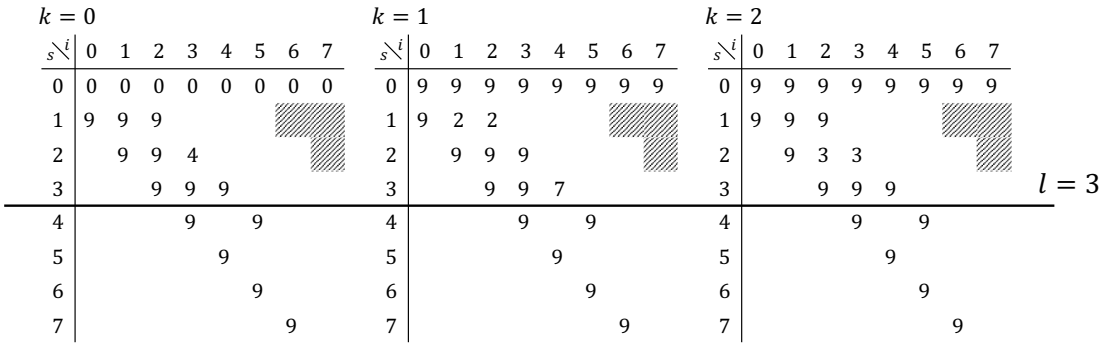

図 4　$A = \texttt{aabacab}, B = \texttt{baabbcaa}, P = \texttt{aab}$ に対してアルゴリズム 1 を動作させた場合の $i = 2$ のループが終了した時点での DP 表の様子.

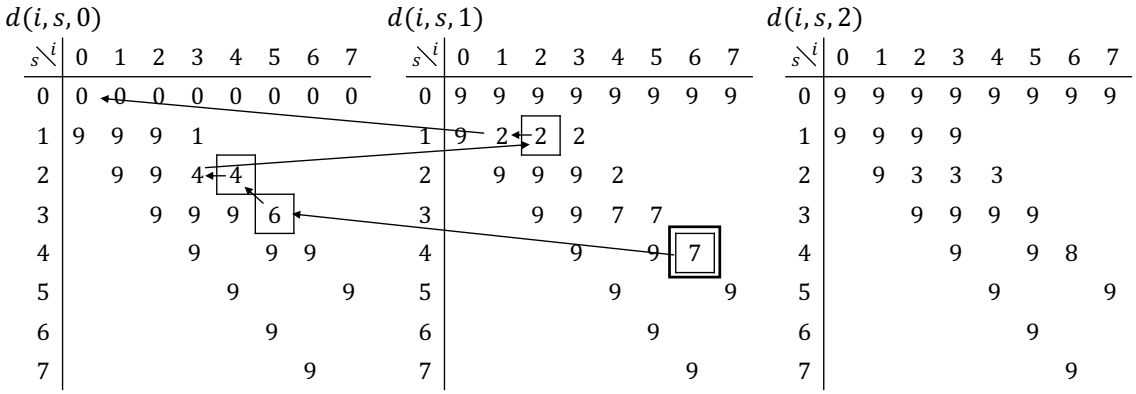

図 5　$A = \texttt{aabacab}, B = \texttt{baabbcaa}, P = \texttt{aab}$ に対する $d(i, s, k)$ の表と $\delta_s(i, s, k)$ と $\delta_k(i, s, k)$ から，STR-EC-LCS のひとつを求める様子.

ムの開発や，Nakatsu らのアルゴリズムを他の GC-LCS 問題群に応用することなどが挙げられる.

参考文献

[1] Nakatsu, N., Kambayashi, Y. and Yajima, S.: A longest common subsequence algorithm suitable for similar text strings, *Acta Informatica*, Vol. 18, No. 2, pp. 171–179 (online), DOI: 10.1007/BF00264437 (1982).

[2] Hirschberg, D. S.: A Linear Space Algorithm for Computing Maximal Common Subsequences, *Commun. ACM*, Vol. 18, No. 6, pp. 341–343 (online), DOI: 10.1145/360825.360861 (1975).

[3] Ullman, J. D., Aho, A. V. and Hirschberg, D. S.: Bounds on the Complexity of the Longest Common Subsequence Problem, *J. ACM*, Vol. 23, No. 1, pp. 1–12 (online), DOI: 10.1145/321921.321922 (1976).

[4] Hirschberg, D. S.: Algorithms for the Longest Common Subsequence Problem, *J. ACM*, Vol. 24, No. 4, pp. 664–675 (online), DOI: 10.1145/322033.322044 (1977).

[5] Sakai, Y.: Computing the Longest Common Subsequence of Two Run-Length Encoded Strings, *Algorithms and Computation* (Chao, K.-M., Hsu, T.-s. and Lee, D.-T., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 197–206 (2012).

[6] Ahsan, S. B., Aziz, S. P. and Rahman, M. S.: Longest common subsequence problem for run-length-encoded strings, *CIT 2012* (2012).

[7] Chen, Y.-C. and Chao, K.-M.: On the generalized constrained longest common subsequence problems, *Journal of Combinatorial Optimization*, Vol. 21, No. 3, pp. 383–392 (online), DOI: 10.1007/s10878-009-9262-5 (2011).

[8] Kuboi, K., Fujishige, Y., Inenaga, S., Bannai, H. and Takeda, M.: Faster STR-IC-LCS Computation via RLE, *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)* (Kärkkäinen, J., Radoszewski, J. and Rytter, W., eds.), Leibniz International Proceedings in Informatics (LIPIcs), Vol. 78, Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 20:1–20:12 (online), DOI: 10.4230/LIPIcs.CPM.2017.20 (2017).

[9] Wang, L., Wang, X., Wu, Y. and Zhu, D.: A dynamic programming solution to a generalized LCS problem, *Information Processing Letters*, Vol. 113, No. 19, pp. 723 – 728 (online), DOI: https://doi.org/10.1016/j.ipl.2013.07.005 (2013).

[10] Liu, J.-J., Wang, Y.-L. and Chiu, Y.-S.: Constrained Longest Common Subsequences with Run-Length-Encoded Strings, *The Computer Journal*, Vol. 58, pp. 1074–1084 (online), DOI: 10.1093/comjnl/bxu012 (2014).

[11] Chen, Y.-C. and Chao, K.-M.: On the generalized constrained longest common subsequence problems, *Journal of Combinatorial Optimization*, Vol. 21, No. 3, pp. 383–392 (online), DOI: 10.1007/s10878-009-9262-5 (2011).