Hardness of longest common subsequence for sequences with bounded run-lengths

Guillaume Blin, Laurent Bulteau, Minghui Jiang, Pedro J. Tejada, Stéphane Vialette

Hardness of longest common subsequence for sequences with bounded run-lengths

Guillaume Blin[1], Laurent Bulteau[2], Minghui Jiang[3], Pedro J. Tejada[3], and Stéphane Vialette[1]

[1] Université Paris-Est, LIGM, UMR 8049, France.
[2] Université de Nantes, LINA, UMR 6241, France.
[3] Utah State University, Department of Computer Science, USA.

Abstract. The longest common subsequence (LCS) problem is a classic and well-studied problem in computer science with extensive applications in diverse areas ranging from spelling error corrections to molecular biology. This paper focuses on LCS for fixed alphabet size and fixed run-lengths (*i.e.*, maximum number of consecutive occurrences of the same symbol). We show that LCS is **NP**-complete even when restricted to (i) alphabets of size 3 and run-length at most 1, and (ii) alphabets of size 2 and run-length at most 2 (both results are tight). For the latter case, we show that the problem is approximable within ratio 3/5.

1 Introduction

The longest common subsequence (lcs for short) problem is a classic and well-studied problem in computer science with extensive applications in diverse areas ranging from spelling error corrections to molecular biology. A *subsequence* of a string is obtained by deleting zero or more symbols of that string. This problem is a specialization of the notion of edit distance in which we do not consider the operation of substitution. Finding the longest string which is equal to a subsequence of two or more strings is known as the *longest common subsequence* (LCS) problem. LCS has been extensively studied during the last 30 years. In particular the case where the number of sequences is 2 has been studied in detail, and LCS is well-known to be polynomial-time solvable by dynamic programming in this case (see [**?**] and references therein). Furthermore, there exist methods with lower complexity which often depend on the length of the lcs, the size of the alphabet, or both (the best general reference is [**?**]). More generally, the problem is solvable in polynomial-time by dynamic programming when the number of input sequences is constant. For the general case of an arbitrary number of input sequences, the problem is **NP**-complete [**?**]. The problem has been also studied in the framework of parameterized complexity [**?**,**?**,**?**]. LCS for unbounded alphabet size is $\mathbf{W}[t]$-hard for $t \geq 1$ when parameterized by the number of input sequences, and $\mathbf{W}[\mathbf{2}]$-hard when parameterized by the length of the sought common subsequence. For a fixed alphabet size, LCS is $\mathbf{W}[\mathbf{1}]$-hard when parameterized by the number of input sequences but is fixed-parameter tractable when parameterized by the length of the sought common subsequence.

Run-length encoding is a well-known method for compressing strings, and a whole line of research is devoted to studying LCS for run-length encoded strings. A string s is

run-length encoded if it is described as an ordered sequence of pairs (σ, i), often denoted σ^i, each consisting of an alphabet symbol σ and an integer i. Each pair corresponds to a *run* in s consisting of i consecutive occurrences of σ. For example, the string $s = bbbbbaaccc$ can be encoded as $b^5a^2c^3$. Two typical examples of run-length encoding are image compression since many images contain large runs of identically-valued pixels, and mini-satellites in biological sequences since these sequences contain a large number of tandem repeats. In this context, two lines of research are being explored. A first line of research has tried to improve the running time of the algorithms by using sparse dynamic programming to compute small subsets of the elements in the standard lcs table [**?**,**?**,**?**,**?**]. A second line of research has tried to find algorithms with running times depending only on the number of runs in the input strings, without computing individual elements of the standard lcs table [**?**,**?**]. It is worth mentioning that work has also been done on computing the similarity of two run-length encoded in the affine gap penalty model [**?**], the string edit distance problem, the pairwise global alignment problem, and the pairwise local alignment problem in the linear-gap model with arbitrary scoring matrices [**?**], and on computing the constrained lcs of run-length encoded strings [**?**]. Refer to [**?**,**?**,**?**,**?**,**?**] and references therein for more problems on run-length encoded strings.

This paper is devoted to studying LCS for the general case of an arbitrary number of input sequences for a fixed size alphabet with a special focus on fixed run-lengths. To shorten notations, for positive integers p and q, we let LCS(p, q) stand for LCS where input sequences are defined over an alphabet of size at most p, and each input sequence has maximum run-length at most q. Abusing notation, we shall write $q = \infty$ to denote unbounded run-lengths.

The paper is organized as follows. In Section 2, we present a new simple proof for the hardness of LCS for binary alphabets and show that LCS$(3, 1)$ is **NP**-complete. Section 3 is devoted to proving hardness of LCS$(2, 2)$, and we consider in Section 4 approximation issues of this problem.

2 Preliminary results and NP-completeness of LCS$(3, 1)$

For an arbitrary number of sequences, Maier [**?**] showed that LCS is **NP**-complete even for an alphabet of size 2 (in our terms, LCS$(2, \infty)$ is **NP**-complete). This result can be found in almost every textbook on algorithms and serves as a classical example to demonstrate the limit of dynamic programming approaches to solving LCS for an arbitrary number of input sequences. However, Maier's proof is notoriously complicated and we propose here as a warm-up an alternate and simpler proof (we shall next adapt this proof to prove the **NP**-completeness of LCS$(3, 1)$).

Proposition 1. LCS$(2, \infty)$ *is* **NP***-complete.*

Proof. Let $G = (V, E)$ be a graph with n vertices and m edges. Write $V = \{1, 2, \ldots, n\}$. We construct $m + 1$ sequences $S_0, S_1, \ldots, S_m$ over alphabet $\Sigma = \{0, 1\}$, each of length at most $(n + 1)^2 - 2$ as follows. The sequence S_0 is defined to be $(0^n1)^n$. For each edge $e_j = \{u, v\} \in E$, $u < v$ and $1 \leq j \leq m$, the sequence S_j is defined to be

$(0^n1)^{u-1}\ 0^n\ (0^n1)^{v-u}\ 0^n\ (0^n1)^{n-v}$. For example, in a graph with 7 vertices the edge between vertices 2 and 4 is represented by the sequence $0^71\ 0^7\ (0^71)^2\ 0^7\ (0^71)^3$.

We claim that the graph G has an independent set of size k if and only if the sequences $S_0, S_1, \ldots, S_m$ have a common subsequence of length $n^2 + k$.

Suppose first that G has an independent set I of size k. Consider the sequence $T = T_1\ T_2\ \ldots\ T_n$, where $T_i = 0^n$ if $i \notin I$ and $T_i = 0^n1$ if $i \in I$. Clearly, $|T| = n^2 + k$, and T is a subsequence of S_0. Furthermore, since each edge has at least one vertex not in I, it can be seen (details omitted) that T is a common subsequence of $S_1, S_2, \ldots, S_m$.

We now prove the reverse implication. Suppose that $S_0, S_1, \ldots, S_m$ have a common subsequence of length $n^2 + k$. Then any lcs of these $m + 1$ input sequences has length at least $n^2 + k$. Let $T_{\mathbf{opt}}$ be an lcs of $S_0, S_1, \ldots, S_m$, and consider a multiple alignment of $S_0, S_1, \ldots, S_m$ inducing $T_{\mathbf{opt}}$. We modify this multiple alignment by shifting 0s right to obtain another multiple alignment inducing $T_{\mathbf{opt}}$ where matched 0s cannot be shifted right anymore. For each sequence S_j, $0 \leq j \leq m$, find the rightmost 0 included in the multiple alignment and shift it right until it is the rightmost 0 of S_j or it is just before a 1 included in the multiple alignment. Observe that each one of those 0s is the rightmost 0 of a 0^n1 or a 0^n substring, and hence all the other 0s in those 0^n1 or 0^n substrings of S_j, $0 \leq j \leq m$, can be aligned by shifting other 0s included in the alignment right (otherwise, $T_{\mathbf{opt}}$ is not an lcs). Moreover, observe that at least one 0 from each 0^n1 substring of S_0 has to be included in the alignment since otherwise we obtain $|T_{\mathbf{opt}}| \leq n(n+1) - n = n^2 < n^2 + k$, for any $k \geq 1$, contradicting our assumption that the length of an lcs is at least $n^2 + k$. Thus by repeating this shifting process for the substrings of S_j, $0 \leq j \leq m$, to the left of the last 0^n1 or 0^n substrings considered, we can make sure that all the 0s in S_0 are included in the alignment, and moreover all the symbols that are included in the alignment from each 0^n1 substring of S_0 are aligned with symbols from the same 0^n1 or 0^n substring of S_j, $1 \leq j \leq m$.

Therefore a longest common subsequence is obtained by including all the 0s in S_0 and maximizing the number of 1s that can be aligned. After the shifting process described above it can be seen that for each edge $e_j = \{u, v\}$ of G, at least one of the two 0^n1 substrings at positions u and v of S_0 must be aligned with a substring 0^n of S_j. To maximize the number of 1s in $T_{\mathbf{opt}}$, pick such vertices to create a minimum vertex cover. The remaining vertices form a maximum independent set and thus $|T_{\mathbf{opt}}| = n^2 + \alpha(G)$. Hence, if we suppose that $S_0, S_1, \ldots, S_m$ has a common subsequence of length $n^2 + k \leq n^2 + \alpha(G)$, then G has an independent set of size $k \leq \alpha(G)$. □

Taking run-lengths into consideration, we observe that LCS(2, 1) is certainly solvable in polynomial-time since each input sequence is a binary string with alternate symbols. (We shall prove in Section 3 that LCS(2, 2) is, however, already **NP**-complete.) The following negative result is thus tight.

Proposition 2. LCS(3, 1) *is* **NP***-complete.*

Proof. Let G be a graph with n vertices and m edges. We construct $m + 1$ sequences of length at most $(n + 1)(2n + 1) - 2$, over alphabet $\Sigma = \{a, b, c\}$. The proof uses

the construction of Proposition 1 where we replace each 0 by ab and each 1 by c. For example, in a graph with 7 vertices the edge between vertices 2 and 4 is represented by the sequence $(ab)^7c\ (ab)^7\ ((ab)^7c)^2\ (ab)^7\ ((ab)^7c)^3$

We claim that the graph G has an independent set of size k if and only if the sequences $S_0, S_1, \ldots, S_m$ have a common subsequence of length $2n^2 + k$.

For the direct implication, the proof works as for the one of Proposition 1. For the reverse implication, a similar argument can be used to show that there is an alignment that induces an lcs including all the as and bs of S_0, and for which all the included symbols from each $(ab)^nc$ substring of S_0 are aligned with symbols from the same $(ab)^nc$ or $(ab)^n$ substring of S_j, $1 \leq j \leq m$: instead of finding the rightmost 0, simply find the rightmost b included in the alignment (if the rightmost symbol in $\{a, b\}$ included in the alignment is an a, the alignment does not induce an lcs) and shift it right until it is the rightmost b of S_j or it is next to a c included in the alignment; then align all the other as and bs in the same $(ab)^nc$ or $(ab)^n$ substrings of S_j, $0 \leq j \leq m$. Then again, an lcs is obtained by including all the as and bs in S_0 and maximizing the number of cs that can be aligned, and the rest of the proof works. □

3 NP-completeness of LCS(2, 2)

In the light of Proposition 1 and Proposition 2, a natural question arises: is LCS(2, 2) polynomial-time solvable? In other words, for an alphabet of size 2, does limiting the run-length to its minimal non-trivial value enough to guarantee tractability? We answer by the negative, and this section is devoted to proving hardness of LCS(2, 2).

The following easy property will turn to be extremely useful for the rest of the discussion as it allows us to focus on common subsequences with run-lengths at most 2.

Proposition 3. *Let $\mathcal{S}$ be an arbitrary input instance of* LCS(2, 2)*. Then, there exists an lcs of $\mathcal{S}$ with maximum run-length* 2*.*

Proof. For any three consecutive identical symbols in a longest common subsequence, the second symbol can always be replaced. □

We, however, observe that the above proposition cannot be taken as a general rule. Indeed, the following two propositions rule out any extension of Proposition 3 to larger alphabets and/or run-lengths.

Proposition 4. *For any integer n, there exist an input instance $\mathcal{S}$ of* LCS(3, 1) *such that every lcs of $\mathcal{S}$ has maximum run-length at least n.*

Proof. It is enough to observe that the lcs of $(01)^n$ and $(02)^n$ is 0^n. □

Proposition 5. *For any integer n, there exist an input instance $\mathcal{S}$ of* LCS(2, 3) *such that every lcs of $\mathcal{S}$ has maximum run-length at least n.*

Proof. Let L_n be the set of all sequences on alphabet $\Sigma = \{0,1\}$ that start with 0, contain exactly n 0s, have run-length at most 3 for 0, and contain no two consecutive 1s (*i.e.*, run-lengths at most 1 for 1). Clearly, a common subsequence of L_n has length n since 0^n is a common subsequence of L_n. We show by induction on n that the only lcs of L_n is 0^n. The property is certainly valid for $1 \leq n \leq 3$ since $0^n \in L_n$. For $n \geq 4$, assume that the property holds up to $n-1$, and suppose, aiming at a contradiction, that there exists a common subsequence T of L_n that contains at least one 1. Write $T = 0^p\,1\,R$, where $0 \leq p \leq n$ and $R \in \Sigma^*$. Define $S = (01)^{p-1}0001$ ($S = 001$ if $p = 0$). The sequence S contains $p+2$ 0s, and we claim that $0 \leq p \leq n-2$. Indeed, if $p > n-2$ then the smallest prefix of S that contains n 0s belongs to L_n but is not a super-sequence of T, a contradiction. Now, define $L' = SL_{n-p-2}$. We have $L' \subseteq L_n$, and hence T is a common subsequence of L'. Furthermore, by construction of S, R is a common subsequence of L_{n-p-2}. Therefore, by the induction hypothesis, R has length at most $n-p-2$, and hence T has length at most $p+1+(n-p-2) = n-1$. □

Before diving into the reduction, we need the following definitions. A 10-*sequence* is a sequence starting with 1 and ending with 0. If S and T are 10-sequences, we say that S is a 10-*tight* subsequence of T if S is a subsequence of T and neither $10S$ nor $S10$ is a subsequence of T. The following easy lemmas are used in upcoming Proposition 6.

Lemma 1. *Let S_1, T_1, S_2, and T_2 be* 10 *sequences. If S_1 is a* 10*-tight subsequence of T_1 and S_2 is a* 10*-tight subsequence of T_2, then S_1S_2 is a* 10*-tight subsequence of T_1T_2.*

Lemma 2. *Let S_1, S_2, S_3, T_1, T_2, T_3 be* 10*-sequences where S_1 is a* 10*-tight subsequence of T_1, and S_3 is a* 10*-tight subsequence of T_3. Then, $S_1S_2S_3$ is a subsequence of $T_1T_2T_3$ if and only if S_2 is a subsequence of T_2.*

Proposition 6. LCS$(2,2)$ *is* **NP***-complete.*

The proof is by a reduction from 3-SAT. Let an arbitrary instance of 3-SAT be given by a CNF formula Φ with n variables $\{v_1,\ v_2, \ldots, v_n\}$ and m clauses. For any variable v_i, we write $+v_i$ for positive literal v_i, $-v_i$ for the negative literal $\overline{v_i}$, and $\pm v_i$ for any literal of variable v_i. First, we define 9 basic substrings as follows (bold is used to emphasize some difference between the strings; A_0 vs B_0, $\{X,Y,Z\}_-$ vs $\{X,Y,Z\}_+$):

$$\begin{array}{lll} A_0 = 1011\,\mathbf{0011}\,0010 & B_0 = 1011\,\mathbf{0101}\,0010 & D_0 = 1100\,1100\,1100 \\ X_- = 1011\,\mathbf{001}\,00100 & Y_- = 11011\,\mathbf{001}\,00100 & Z_- = 11011\,\mathbf{001}\,0010 \\ X_+ = 1011\,\mathbf{011}\,00100 & Y_+ = 11011\,\mathbf{011}\,00100 & Z_+ = 11011\,\mathbf{011}\,0010 \end{array}$$

We now define $m+2$ sequences $\{A, B, C_1, C_2, \ldots, C_m\}$ based on these 9 substrings. The first two sequences are simply defined to be $A = (A_0)^n$ and $B = (B_0)^n$. For each $1 \leq j \leq m$, write $x \vee y \vee z$ for the j-th clause of the formula, where literals x, y and z are $\pm v_a$, $\pm v_b$, and $\pm v_c$ respectively and $a < b < c$. Define

$$X_j = \begin{cases} X_- \text{ if } x = -v_a \\ X_+ \text{ if } x = +v_a \end{cases} \qquad Y_j = \begin{cases} Y_- \text{ if } y = -v_b \\ Y_+ \text{ if } y = +v_b \end{cases} \qquad Z_j = \begin{cases} Z_- \text{ if } z = -v_c \\ Z_+ \text{ if } z = +v_c \end{cases}$$

and

$$\mathcal{L} = (A_0)^{a-1}\, X_j\, (1\,0\,D_0)^{b-a-1} \qquad \mathcal{R} = (D_0\,1\,0)^{c-b-1}\, Z_j\, (A_0)^{n-c}.$$

The string C_j is defined to be $C_j = \mathcal{L}\,1\,0\,Y_j\,1\,0\,\mathcal{R}$.

After the construction step, we now turn to proving the correctness of the reduction. We need some technical lemmas. Let $P = 1011\,\mathbf{011}\,0010$ and $N = 1011\,\mathbf{001}\,0010$.

Lemma 3. *A and B have 2^n lcs, they are exactly $\{P, N\}^n$.*

Proof. Consider an alignment that induces an lcs of A and B. Note that B_0 is divided into a left block 101101 and a right block 010010. There are $2n$ such blocks in B. We show that the number of fully matched blocks is exactly $m = n$.

Every sequence of $\{P, N\}^n$ has length $11n$ and is a common subsequences of A and B. Since $|A| = |B| = 12n$, both A and B have n symbols that are not part of the common subsequence. Therefore, at most n blocks can be partially matched (otherwise we miss more than n symbols), and hence $m \geq n$. Furthermore, each block of B that is fully matched introduces at least one gap in A (since no block of B is a substring of A). Thus we also have $m \leq n$. Moreover, in A, the gap between two substrings matched to two consecutive blocks of B must be 0.

In the alignment, the $2n$ blocks of B specify $2n$ corresponding blocks of A, where each fully matched block of B corresponds to a partially matched block of A with one gap, and, each partially matched block of B with one gap corresponds to a completely matched block of A. Observe that a completely matched block (10110)1 of B introduces a single gap in A only if it corresponds to a partially matched block (10110)01 of A which is a prefix of A_0, and that a completely matched block 0(10010) of B introduces a single gap in A only if it corresponds to a partially matched block 01(10010) of A which is a suffix of A_0. Since the prefix (10110)01 and the suffix 01(10010) both overlap in each A_0, there must be at most one completely matched block (10110)1 or 0(10010) in each B_0. Hence each B_0 contains one fully matched block and one partially matched block, and, for each B_0, the lcs contains either (10110)1 (10010) $= P$ or (10110) 0(10010) $= N$. Therefore, the lcs is in $\{P, N\}^n$. □

According to Lemma 3, if sequences $\{A, B, C_1, C_2, \ldots, C_m\}$ have an lcs of length $11n$, then it is in $\{P, N\}^n$. A sequence $S \in \{P, N\}^n$ is easily mapped to a truth assignment ϕ_S as follows: $\phi_S(v_i) =$ true if the i-th block of S is P, and $\phi_S(v_i) =$ false otherwise.

Lemma 4. *For any j and any sequence $S \in \{P, N\}^n$, S is a subsequence of C_j if and only if $\phi(v_s)$ satisfies the j-th clause of Φ.*

Proof. We write S_i for the i-th block of S (such that for every i, $S_i \in \{P, N\}$). For $i \leq j$, we write $S_{i..j} = S_i S_{i+1} \ldots S_j$. If ϕ_S satisfies a boolean term t, we write $\phi_S \vDash t$, otherwise $\phi_S \nvDash t$. We first need to compare P and N with each basic substring of C_j, and we obtain the following important relations (the proof is tedious but easy).

- P is a 10-tight subsequence of A_0, $10D_0$, D_010, X_+, Z_+, X_-10, $10Z_-$.
- N is a 10-tight subsequence of A_0, $10D_0$, D_010, X_-, Z_-, X_+10, $10Z_+$.

– P (but not N) is a subsequence of Y_+, and N (but not P) is a subsequence of Y_-. P and N are subsequences of $10Y_-$, Y_-10, $10Y_+$ and Y_+10.

We prove the lemma by combining Lemma 1 with the above relations. The proof is divided into three parts.

Part 1: from 1 **to** $b-1$. Each S_i, $1 \leq i < a$, is a 10-tight subsequence of A_0, and hence $S_{1..a-1}$ is a 10-tight subsequence of $(A_0)^{a-1}$. Furthermore, sequence S_a is a subsequence of X_j if and only if $\phi_S \vDash x$ (since P is a subsequence of X_+ but not of X_-, and N is a subsequence of X_- but not of X_+). Similarly, S_a is a 10-tight subsequence of X_j10 if and only if $\phi_S \nvDash x$. Hence,

– if $\phi_S \vDash x$, $S_{1..a}$ is a subsequence of $(A_0)^{a-1}X_j$,
– if $\phi_S \nvDash x$, $S_{1..a}$ is a 10-tight subsequence of $(A_0)^{a-1}X_j10$.

Moreover, each S_i, $a < i < b$, is a 10-tight subsequence of $10D_0$ and D_010, and hence

– if $\phi_S \vDash x$, $S_{1..b-1}$ is a subsequence of $(A_0)^{a-1}X_j(10D_0)^{b-a-1} = \mathcal{L}$,
– if $\phi_S \nvDash x$, $S_{1..b-1}$ is a 10-tight subsequence of $(A_0)^{a-1}X_j(10D_0)^{b-a-1}10 = \mathcal{L}10$.

Part 2: from n **down to** $b+1$. Using a similar argument as in Part 1, reading sequences from right to left, with Z_j instead of X_j, we have:

– if $\phi_S \vDash z$, $S_{b+1..n}$ is a subsequence of $(D_010)^{c-b-1}Z_j(A_0)^{n-c} = \mathcal{R}$,
– if $\phi_S \nvDash z$, $S_{b+1..n}$ is a 10-tight subsequence of $10(D_010)^{c-b-1}Z_j(A_0)^{n-c} = 10\mathcal{R}$.

Part 3: junction. If $\phi_S \vDash x$, then $S_{1..b-1}$, S_b, $S_{b+1..n}$ are subsequences of $\mathcal{L}$, $10Y_j$, $10\mathcal{R}$, respectively, and hence S is a subsequence of $C_j = \mathcal{L}10Y_j10\mathcal{R}$. If $\phi_S \vDash y$, then $S_{1..b-1}$, S_b, $S_{b+1..n}$ are subsequences of $\mathcal{L}10$, Y_j, $10\mathcal{R}$, respectively, and hence S is a subsequence of $C_j = \mathcal{L}10Y_j10\mathcal{R}$. If $\phi_S \vDash z$, then $S_{1..b-1}$, S_b, $S_{b+1..n}$ are subsequences of $\mathcal{L}10$, Y_j10, $\mathcal{R}$, respectively, and hence S is a subsequence of $C_j = \mathcal{L}10Y_j10\mathcal{R}$.

If $\phi_S \nvDash x \vee y \vee z$, then S_b is not a subsequence of Y_j. Since $S_{1..b-1}$ and $S_{b+1..n}$ are 10-tight subsequences of $\mathcal{L}10$ and $10\mathcal{R}$, respectively, then, according to Lemma 2, $S = S_{1..b-1}S_bS_{b+1..n}$ is not a subsequence of $C_j = \mathcal{L}10\ Y_j\ 10\mathcal{R}$. □

Proof (Proof of Proposition 6.). Let S be an lcs of $\{A, B, C_1, \ldots, C_m\}$. If S has length at least $11n$, then by Lemma 3 it is in $\{P, N\}^n$, and by Lemma 4, ϕ_S satisfies every clause of Φ, and hence Φ is satisfiable. Conversely, if Φ is satisfiable, set any truth assignment, and create sequence $S = S_1\, S_2\, \ldots\, S_n$ such that $S_i = P$ if v_i is true, and $S_i = N$ otherwise. Then S is a common subsequence of $\{A, B, C_1, C_2, \ldots, C_m\}$ of size $11n$. Since the construction of sequences $A, B, C_1, \ldots, C_m$ can be carried on in polynomial-time, LCS$(2, 2)$ is **NP**-complete. □

4 Approximation

LCS is approximable within ratio $O(m/\log(m))$, where m is the length of the shortest input string [**?**]. It is, however, not approximable within ratio n^ε for any constant $\varepsilon < 1$,

where n is the length of the longest input string [**?**] (see also [**?**]), and **APX**-hard if the size of the alphabet is fixed [**?**]. Despite the discouraging results, it has been proved that LCS over a fixed alphabet can be indeed very well approximated on the average by using a simple algorithm called *Long Run* [**?**]. Bonizzoni et al. [**?**] developed an approximate algorithm for LCS called *Expansion Algorithm* (their algorithm first compresses sequences to streams by the same concept of run-length encoding, then progressively find a common sequence of all streams by the bottom-up tree merging technique).

We present here a 3/5-approximation algorithm for LCS$(2, 2)$. The algorithm is by combining an exhaustive search for a limited number of common subsequences with linear programming techniques.

Proposition 7. LCS$(2, 2)$ *is approximable within ratio* $3/5$.

Proof. Let $S_1, S_2, \ldots, S_m$, be m input sequences over alphabet $\Sigma = \{0, 1\}$ with maximum run-length 2. The approximation algorithm is as follows:

Input: Subsequences $S_1, S_2, \ldots, S_n$ over alphabet $\Sigma = \{0, 1\}$
Output: A common subsequence of $S_1, S_2, \ldots, S_n$

for all $P \in \{0, 1, 01, 001, 011, 0011\}$ **do**
  Let T_P be the lcs of $S_1, S_2, \ldots, S_n$ in P^*
end for
return the length of the longest common subsequence T_P

For the sake of clarity, write $A = 01$, $B = 001$, $C = 011$, $D = 0011$, and $X = \{A, B, C, D\}$. We focus on the case where each input sequence starts with 0 and terminates with 1 (the general case is easily deduced from this restriction).

Let $T_{\mathbf{opt}}$ be an lcs of $S_1, S_2, \ldots, S_m$. According to Proposition 3, there is no loss of generality in assuming that $T_{\mathbf{opt}}$ has maximum run-length 2. Let $n_{\mathbf{opt}} = |T_{\mathbf{opt}}|$, and k be the number of 01 substrings in $T_{\mathbf{opt}}$. First, it is easily seen that X is a code (*i.e.*, any $u \in \Sigma^*$ has at most one X-factorization). Furthermore, since each input sequence starts with 0 and terminates with 1, so does any lcs, and hence $T_{\mathbf{opt}}$. Let $T_{\mathbf{opt}} = t_1 t_2 \ldots t_k$ be the X-factorization of $T_{\mathbf{opt}}$. Let n_A (respectively n_B, n_C, and n_D) be the number of indices i, $1 \leq i \leq k$, such that $t_i = A$ (respectively, $t_i = B$, $t_i = C$, and $t_i = D$), and define $\overline{n_A} = n_A/n_{\mathbf{opt}}$, $\overline{n_B} = n_B/n_{\mathbf{opt}}$, $\overline{n_C} = n_C/n_{\mathbf{opt}}$, $\overline{n_D} = n_D/n_{\mathbf{opt}}$. Then it follows that $n_{\mathbf{opt}} = 2n_A + 3n_B + 3n_C + 4n_D$, and hence

$$2\overline{n_A} + 3\overline{n_B} + 3\overline{n_C} + 4\overline{n_D} \geq 1 \tag{1}$$

(Notice that (1) is fundamentally an equality but only the "$\geq$" part is used in the rest of the proof.)

We now turn to relating $|T_P|$, $P \in \{0, 1, 01, 001, 011, 0011\}$, to n_A, n_B, n_C, and n_D. From optimality of $|T_P|$, $P \in \{0, 1, 01, 001, 011, 0011\}$, we obtain

$$\begin{aligned}
|T_0| &\geq n_A + 2n_B + n_C + 2n_D \\
|T_1| &\geq n_A + n_B + 2n_C + 2n_D \\
|T_A| &\geq 2n_A + 2n_B + 2n_C + 2n_D \\
|T_B| &\geq 3n_B + 3n_D \\
|T_C| &\geq 3n_C + 3n_D \\
|T_D| &\geq 4n_D.
\end{aligned}$$

By definition, the approximation ratio r of our algorithm is defined by

$$r = \max\{|T_0|, |T_1|, |T_A|, |T_B|, |T_C|, |T_D|\}/n_{\mathbf{opt}},$$

and hence

$$r \geq \overline{n_A} + 2\overline{n_B} + \overline{n_C} + 2\overline{n_D} \tag{2}$$
$$r \geq \overline{n_A} + \overline{n_B} + 2\overline{n_C} + 2\overline{n_D} \tag{3}$$
$$r \geq 2\overline{n_A} + 2\overline{n_B} + 2\overline{n_C} + 2\overline{n_D} \tag{4}$$
$$r \geq 3\overline{n_B} + 3\overline{n_D} \tag{5}$$
$$r \geq 3\overline{n_C} + 3\overline{n_D} \tag{6}$$
$$r \geq 4\overline{n_D} \tag{7}$$

Inequalities (1) to (7) together with domain constraints $n_A \geq 0$, $n_B \geq 0$, $n_C \geq 0$, and $n_D \geq 0$ define a (minimization) linear program (LP) that can be solved to optimality in polynomial-time. Let r^* be the optimal solution of the defined LP (referred to as the primal problem). The dual LP reads as follows:

$$\begin{aligned}
&\text{maximize } y_1 \\
&\text{such that } y_2 + y_3 + y_4 + y_5 + y_6 + y_7 \leq 1 \\
&\qquad 2y_1 \leq y_2 + y_3 + 2y_4 \\
&\qquad 3y_1 \leq 2y_2 + y_3 + 2y_4 + 3y_5 \\
&\qquad 3y_1 \leq y_2 + 2y_3 + 2y_4 + 3y_6 \\
&\qquad 4y_1 \leq 2y_2 + 2y_3 + 2y_4 + 3y_5 + 3y_6 + 4y_7
\end{aligned}$$

By the strong duality theorem, r^* is also an optimal solution for the dual LP. A lower bound for the dual LP is obtained as follows: $y_2 = y_3 = y_7 = 0$, $y_4 = 3/5$, $y_5 = y_6 = 1/5$, and $y_1 = 3/5$, and hence $r^* \geq 3/5$, thereby proving the proposition. □

Two remarks are worth making about the proof of Proposition 7. First, the approximation ratio of the proposed algorithm is exactly 3/5. Indeed, for the sequence $(0110010011)^n$ none of the tested patterns in $\{0, 1, 01, 001, 011, 0011\}$ would return a subsequence of length greater than $3n/5$. Second, our lower bound for the dual problem is defined by $y_2 = y_3 = y_7 = 0$, and hence using patterns A^*, B^* and C^* only would yield the same approximation ratio.

References

1. H.-Y. Ann, C.-B. Yang, C.-T. Tseng, and C.-Y.Hor. Fast algorithms for computing the constrained lcs of run-length encoded strings. In H.R. Arabnia and M.Q. Yang, editors, *Proc. International Conference on Bioinformatics & Computational Biology (BIOCOMP), Las Vegas, USA*, CSREA Press, pages 646–649, 2009.
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*, 108:360–364, 2008.
3. A. Apostolico, G.M. Landau, and S. Skiena. Matching for run-length encoded strings. *Journal of Complexity*, 15(1):4–16, 1999.
4. L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *Proc. of the 7th International Symposium on String Processing Information Retrieval (SPIRE), Coruña, Spain*, pages 39–48. IEEE Computer Society, 2000.
5. P. Berman and G. Schnitger. On the complexity of approximating the independent set problem. *Information and Computation*, 96:77–94, 1992.
6. H.L. Bodlaender, R.G. Downey, M.R. Fellows, M.T. Hallett, and H.T. Wareham. Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences*, 11(1):49–57, 1995.
7. H.L. Bodlaender, R.G. Downey, M.R. Fellows, and H.T. Wareham. The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science*, 147:31–54, 1994.
8. P. Bonizzoni, G. Della Vedova, and G. Mauri. Experimenting an approximation algorithm for the lcs. *Discrete Applied Mathematics*, 110(1):13–24, 2001.
9. H. Bunke and J. Csirik. An improved algorithm for computing the edit distance of run-length coded strings. *Information Processing Letters*, 54:93–96, 1995.
10. M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge, 2007.
11. V. Freschi and A. Bogliolo. Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism. *Information Processing Letters*, 90:167–173, 2004.
12. M.M. Halldórsson. Approximation via partitioning. Technical report, School of Information Science, Japan Advanced Institute of Science and Technology, Hokuriku, 1995.
13. P.-H. Hsu, K.-Y. Chen, , and K.-M. Chao. Finding all approximate gapped palindromes. In Y. Dong, D.-Z. Du, and O.H. Ibarra, editors, *Proc. 20th International Symposium Algorithms and Computation (ISAAC), Honolulu, USA*, number 5878 in Lecture Notes in Computer Science, pages 1084–1093, 2009.
14. G.S. Huang, J.J. Liu, and Y.L. Wang. Sequence alignment algorithms for run-length-encoded strings. In Xi. Hu and J. Wang, editors, *Proc. 14th Annual International Computing and Combinatorics Conference (COCOON), Dalian, China*, number 5092 in Lecture Notes in Computer Science, pages 319–330, 2008.
15. T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal on Computing*, 24:1122–1139, 1995.
16. P.-H. Hsu K.-Y. Chen and K.-M. Chao. Finding all approximate gapped palindromes. In O. Cheong, K.-Y Chwa, and K Park, editors, *Proc. 21th International Symposium Algorithms and Computation (ISAAC), Jeju Island, Korea*, number 6507 in Lecture Notes in Computer Science, pages 339–350, 2010.
17. J.W. Kim, A. Amir, G.M. Landau, and K. Park. Computing similarity of run-length encoded strings with affine gap penalty. *Theoretical Computer Science*, 395:268–282, 2008.
18. J.J. Liu, G.S. Huang, and Y.L. Wang. A fast algorithm for finding the positions of all squares in a run-length encoded string. *Theoretical Computer Science*, 410:3942–3948, 2009.
19. J.J. Liu, G.S. Huang, Y.L. Wang, and R.C.T. Lee. Edit distance for a run-length-encoded string and an uncompressed string. *Information Processing Letters*, 105:12–16, 2007.
20. 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*, 24:173–184, 2008.
21. D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2):322–336, 1978.

22. W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, and K. Hashimoto. Efficient algorithms to compute compressed longest common substrings and compressed palindromes. *Theoretical Computer Science*, 410:900–913, 2009.
23. J.S.B. Mitchell. A geometric shortest path problem, with application to computing a longest common subsequence in run-length encoded strings. Technical report, Department of Applied Mathematics, SUNY Stony Brook, 1997.
24. K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. of Computer and System Sciences*, 67(4):757–771, 2003. Special issue on Parameterized computation and complexity.