Journal of Computer and System Sciences 78 (2012) 689–697

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Approximability of constrained LCS ☆

Minghui Jiang

Department of Computer Science, Utah State University, Logan, UT 84322, USA

ARTICLE INFO

Article history:
Received 17 January 2011
Received in revised form 25 September 2011
Accepted 10 October 2011
Available online 20 October 2011

Keywords:
Longest common subsequence
Computational biology
Computational complexity
Approximation algorithms

ABSTRACT

The problem Constrained Longest Common Subsequence is a natural extension to the classical problem Longest Common Subsequence, and has important applications to bioinformatics. Given k input sequences $A_1, \ldots, A_k$ and l constraint sequences $B_1, \ldots, B_l$, C-LCS(k, l) is the problem of finding a longest common subsequence of $A_1, \ldots, A_k$ that is also a common supersequence of $B_1, \ldots, B_l$. Gotthilf et al. gave a polynomial-time algorithm that approximates C-LCS$(k, 1)$ within a factor $\sqrt{\hat{m}|\Sigma|}$, where $\hat{m}$ is the length of the shortest input sequence and $|\Sigma|$ is the alphabet size. They asked whether there are better approximation algorithms and whether there exists a lower bound. In this paper, we answer their questions by showing that their approximation factor $\sqrt{\hat{m}|\Sigma|}$ is in fact already very close to optimal although a small improvement is still possible:

1. For any computable function f and any $\epsilon > 0$, there is no polynomial-time algorithm that approximates C-LCS$(k, 1)$ within a factor $f(|\Sigma|) \cdot \hat{m}^{1/2-\epsilon}$ unless NP = P. Moreover, this holds even if the constraint sequence is unary.
2. There is a polynomial-time randomized algorithm that approximates C-LCS$(k, 1)$ within a factor $|\Sigma| \cdot O(\sqrt{\mathrm{OPT} \cdot \log\log \mathrm{OPT} / \log \mathrm{OPT}})$ with high probability, where OPT is the length of the optimal solution, $\mathrm{OPT} \leqslant \hat{m}$.

For the problem over an alphabet of arbitrary size, we show that

3. For any $\epsilon > 0$, there is no polynomial-time algorithm that approximates C-LCS$(k, 1)$ within a factor $\hat{m}^{1-\epsilon}$ unless NP = P.
4. There is a polynomial-time algorithm that approximates C-LCS$(k, 1)$ within a factor $O(\hat{m}/\log \hat{m})$.

We also present some complementary results on exact and parameterized algorithms for C-LCS$(k, 1)$.

1. Introduction

Longest Common Subsequence (LCS) is a fundamental problem in computer science. Given two input sequences A_1 and A_2 and one constraint sequence B, Constrained Longest Common Subsequence (C-LCS) is the problem of finding a longest common subsequence of A_1 and A_2 that is also a supersequence of B. The problem C-LCS is a natural extension to the classical problem LCS, and has application to computing the homology of two biological sequences with a specific or putative

☆ Supported in part by NSF grant DBI-0743670. A preliminary version appeared in the Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC 2010), part II, pp. 180–191.

E-mail address: mjiang@cc.usu.edu.

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2011.10.002

structure in common [14]. The two biological sequences under examination could be two DNA sequences sharing common sequence motifs with variable gaps, two protein sequences sharing common structural motifs, two RNA sequences sharing common secondary structures, or two genomes sharing common synteny blocks. For example, given two input sequences $A_1 = \text{cgattggcgcactgccaacata}$ and $A_2 = \text{gtattggccgcatgccaaatta}$ and a constraint sequence $B = \textbf{ttggcgccaa}$ (derived from the NF-I transcription factor 5′-TTGGCXXXXXGCCAA-3′), the sequence $C = \text{ga}\textbf{ttggc}\text{gcat}\textbf{gccaa}\text{ata}$ is a longest common subsequence of A_1 and A_2 that is also a supersequence of B.

We review some standard definitions and notations. Given two sequences S and T, we say that S is a *subsequence* of T if S can be obtained from T by deleting zero or more letters without changing the order of the remaining letters, and we say that T is a *supersequence* of S if S is a subsequence of T. For a sequence S, let $S[i]$ denote the letter of S at position i, let $S[i, j]$ denote the subsequence of S starting at position i and ending at position j (the subsequence is empty when $i > j$), and let $|S|$ denote the length of S. For two sequences S and T, let ST denote the concatenation of S and T, and write $S \preccurlyeq T$ if S is a subsequence of T. For a sequence R and a non-negative integer r, let R^r denote a sequence consisting of r repetitions of R concatenated together.

The problem C-LCS can be easily generalized to a problem C-LCS(k, l) for an arbitrary number k of input sequences and an arbitrary number l of constraint sequences [6]:

Problem C-LCS(k, l)

Instance: k input sequences $A_1, \ldots, A_k$ and l constraint sequences $B_1, \ldots, B_l$ over an alphabet Σ, where $k \geqslant 2$, $l \geqslant 1$, and $|\Sigma| \geqslant 2$.
Problem: Find a longest sequence C such that $C \preccurlyeq A_i$ for each i, $1 \leqslant i \leqslant k$, and $B_j \preccurlyeq C$ for each j, $1 \leqslant j \leqslant l$.

Here the input size n is the total length of the k input sequences and the l constraint sequences.

For C-LCS$(2, 1)$, the most basic version of the problem C-LCS on two input sequences A_1 and A_2 and one constraint sequence B, there are dynamic programming algorithms running in $O(|A_1| \cdot |A_2| \cdot |B|)$ time [2,5]; see also [9,1,4,3] for some related results. The problem C-LCS(k, l) becomes intractable, however, when either the number k of input sequences or the number l of constraint sequences is unbounded.

An early result of Middendorf [12] on consistent sequences of type (Super, Sub) implies that even if the input and constraint sequences are over a binary alphabet, it is already NP-hard to decide whether a given instance of C-LCS$(2, l)$ has a valid solution; see also [13]. Recently, Gotthilf et al. [6] showed that if the sequences are over an arbitrary alphabet, then even if all constraint sequences have length 1, it is again NP-hard to decide whether a given instance of C-LCS$(2, l)$ has a valid solution. On the other hand, Gotthilf et al. [6] observed that C-LCS$(k, 1)$ is NP-hard because it generalizes the classical problem LCS on an arbitrary number k of input sequences, which is known to be NP-hard even if the input sequences are over a binary alphabet [11].

C-LCS$(k, 1)$ is perhaps the most interesting variant of C-LCS because of its biological applications, hence it will be the focus of this paper. Let $A_1, \ldots, A_k$ be the k input sequences, and B be the single constraint sequence. Without loss of generality, we assume that the constraint sequence B has length at least one and is a common subsequence of the k input sequences $A_1, \ldots, A_k$. Put $\hat{m} = \min_{1 \leqslant i \leqslant k} |A_i|$ and $b = |B|$. Then $\hat{m} \geqslant b \geqslant 1$.

Gotthilf et al. [6] gave a polynomial-time algorithm that approximates C-LCS$(k, 1)$ within a factor $\sqrt{\hat{m}|\Sigma|}$, and asked whether there are better approximation algorithms and whether there exists a lower bound. In the following two theorems, we show that their approximation factor $\sqrt{\hat{m}|\Sigma|}$ is in fact already very close to optimal but nevertheless a small improvement is still possible:

Theorem 1. *For any computable function f and any $\epsilon > 0$, there is no polynomial-time algorithm that approximates* C-LCS$(k, 1)$ *within a factor $f(|\Sigma|) \cdot \hat{m}^{1/2-\epsilon}$ unless* NP = P. *Moreover, this holds even if the constraint sequence is unary.*

Theorem 2. *There is a polynomial-time randomized algorithm that approximates* C-LCS$(k, 1)$ *within a factor $|\Sigma| \cdot O(\sqrt{\text{OPT} \cdot \log\log \text{OPT} / \log \text{OPT}})$ with high probability, where* OPT *is the length of the optimal solution,* $\text{OPT} \leqslant \hat{m}$.

For an alphabet of arbitrary size, we can have $|\Sigma| = \Theta(\hat{m})$. Then the approximation factor of Gotthilf et al.'s algorithm [6] becomes $\sqrt{\hat{m}|\Sigma|} = \Theta(\hat{m})$. In the following two theorems, we show that again this approximation factor $\Theta(\hat{m})$ is very close to optimal but nevertheless a small improvement is possible:

Theorem 3. *For any $\epsilon > 0$, there is no polynomial-time algorithm that approximates* C-LCS$(k, 1)$ *within a factor $\hat{m}^{1-\epsilon}$ unless* NP = P.

Theorem 4. *There is a polynomial-time algorithm that approximates* C-LCS$(k, 1)$ *within a factor $O(\hat{m} / \log \hat{m})$.*

Although the focus of this paper is on approximability, we also obtain some complementary results on exact and parameterized algorithms. The following theorem shows that C-LCS(k, l) is fixed-parameter tractable with both the alphabet size $|\Sigma|$ and the optimal solution length OPT as parameters, and is polynomially solvable if both k and l are constants:

Theorem 5. C-LCS(k, l) *admits an exact algorithm running in time* $O(|\Sigma|^{\mathrm{OPT}+1} \cdot n)$, *where* OPT *is the length of the optimal solution, and admits an exact algorithm running in time* $O(\prod_{i=1}^{k}(|A_i|+1) \cdot \prod_{j=1}^{l}(|B_j|+1) \cdot (k+l))$.

2. Approximation lower bounds for C-LCS(k, 1)

In this section we prove Theorems 1 and 3.

2.1. Proof of Theorem 1

We prove the inapproximability of C-LCS$(k, 1)$ by a reduction from Max-Clique. Our construction is inspired by Middendorf [12, Theorem 2(b)].

Let G be a graph with n vertices and m edges. We construct a C-LCS$(k, 1)$ instance consisting of

$$k = \binom{n}{2} - m + 1$$

input sequences $A_1, \ldots, A_k$ over a binary alphabet and a single constraint sequence B.

The constraint sequence B is a unary sequence of $n-1$ zeros:

$$B = 0^{n-1}.$$

The last input sequence A_k consists of n^2 ones and $n-1$ zeros:

$$A_k = 1^n \left(01^n\right)^{n-1}.$$

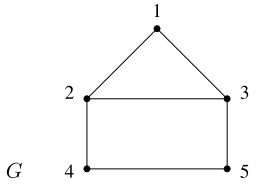
Let $V = \{1, \ldots, n\}$ be the set of vertices of the graph G. Let $\bar{E}$ be the $\binom{n}{2} - m$ pairs of vertices of the graph G that are not edges. For each pair of vertices $\bar{e}_j = \{u, v\} \in \bar{E}$, $1 \leqslant j \leqslant \binom{n}{2} - m$ and $1 \leqslant u < v \leqslant n$, we construct a corresponding input sequence A_j of $n^2 - n$ ones and n zeros:

$$A_j = \left(1^n 0\right)^{u-1} 0 \left(1^n 0\right)^{v-u} \left(01^n\right)^{n-v}.$$

For comparison, observe that

$$A_k = \left(1^n 0\right)^{u-1} 1^n \left(01^n\right)^{v-u} \left(01^n\right)^{n-v}.$$

We use the term *one-block* to refer to a substring of n consecutive ones in the input sequences. Note that each input sequence A_j for $1 \leqslant j \leqslant \binom{n}{2} - m$ consists of $n-1$ one-blocks and n zeros, while the last input sequence A_k consists of n one-blocks and $n-1$ zeros. Thus $\hat{m} = (n-1)n + n = n^2$. This completes the construction. We refer to Fig. 1 for an example.



$$\begin{array}{lcl}
A_1 & = & 0\ 11111\ 0\ 11111\ 0\ 11111\ 0\ 0\ 11111 \\
A_2 & = & 0\ 11111\ 0\ 11111\ 0\ 11111\ 0\ 11111\ 0 \\
A_3 & = & 11111\ 0\ 0\ 11111\ 0\ 11111\ 0\ 11111\ 0 \\
A_4 & = & 11111\ 0\ 11111\ 0\ 0\ 11111\ 0\ 0\ 11111 \\
A_5 & = & 11111\ 0\ 11111\ 0\ 11111\ 0\ 11111\ 0\ 11111 \\
B & = & 0\ 0\ 0\ 0 \\
C & = & 11111\ 0\ 11111\ 0\ 11111\ 0\ 0
\end{array}$$

Fig. 1. A graph G with $n = 5$ vertices and $m = 6$ edges. The C-LCS$(k, 1)$ instance of $k = \binom{5}{2} - 6 + 1 = 5$ input sequences $A_1, \ldots, A_5$ and a single constraint sequence B. The four input sequences A_1, A_2, A_3, A_4 correspond to the 4 non-edges $\{1, 4\}, \{1, 5\}, \{2, 5\}, \{3, 4\}$. The sequence C corresponds to the clique $\{1, 2, 3\}$.

Lemma 1. *There is a clique* K *of* q *vertices in the graph* G *if and only if there is a sequence* C *of length* $\ell = (q+1)n - 1$ *that is a subsequence of each input sequence* A_i *and is a supersequence of the constraint sequence* B.

Proof. We first prove the direct implication. Suppose there is a clique K of q vertices in the graph G. Let $i_1, \ldots, i_q$ be the q vertices in K, where $1 \leqslant i_1 < \cdots < i_q \leqslant n$. We construct a sequence C of qn ones and $n-1$ zeros as follows:

$$C = 0^{i_1-1}1^n 0^{i_2-i_1}1^n \ldots 0^{i_q-i_{q-1}}1^n 0^{n-i_q}.$$

Clearly, C is a supersequence of B. Note that C can be obtained from A_k by deleting all one-blocks except those with indices $i_1, \ldots, i_q$. So C is a subsequence of A_k. For each input sequence A_j, $1 \leqslant j \leqslant \binom{n}{2} - m$, the two vertices u and v of the corresponding non-edge $\bar{e}_j$ cannot be both in the clique K. Consider two cases:

1. If $u \notin K$, then C is a subsequence of the following common subsequence of A_j and A_k

   $$\left(1^n 0\right)^{u-1} \left(01^n\right)^{v-u} \left(01^n\right)^{n-v},$$

   which can be obtained either from A_j by deleting the zero with index v, or from A_k by deleting the one-block with index u.
2. If $v \notin K$, then C is a subsequence of the following common subsequence of A_j and A_k

   $$\left(1^n 0\right)^{u-1} \left(1^n 0\right)^{v-u} \left(01^n\right)^{n-v},$$

   which can be obtained either from A_j by deleting the zero with index u, or from A_k by deleting the one-block with index v.

In either case, C is a subsequence of A_j.

We next prove the reverse implication. Suppose there is sequence C of length $\ell = (q+1)n - 1$ that is a subsequence of each input sequence A_i and is a supersequence of the constraint sequence B. Then C must contain exactly qn ones and exactly $n-1$ zeros, because A_k and B have the same number $n-1$ of zeros. Note that the ones in each input sequence are grouped into one-blocks. When selecting the common subsequence C from each input sequence, we can select the ones from left to right in each one-block and add the remaining ones of a one-block if it is only partially selected. In this way, we obtain a sequence C' that is a supersequence of C and is still a subsequence of each input sequence. Moreover, C' consists of at least q one-blocks and exactly $n-1$ zeros. Let K be the set of vertices corresponding to the indices of these one-blocks in A_k. We claim that K is a clique in the graph G, that is, for each non-edge $\bar{e}_j = \{u, v\}$, either u or v is not in K.

We prove this claim by contradiction. Suppose that both vertices u and v of some non-edge $\bar{e}_j$ are in K. Then the corresponding one-blocks with indices u and v in A_k are selected in C'. Since all $n-1$ zeros in A_k are selected in C', C' contains exactly $u-1$ zeros before the one-block u, exactly $n-v$ zeros after the one-block v, and exactly $v-u$ zeros between them. Observe that for any two one-blocks in A_j, if there are at least $u-1$ zeros before the left one-block and there are at least $n-v$ zeros after the right one-block, then these two one-blocks must both come from the middle part $0(1^n0)^{v-u}$ of A_j, and hence have at most $v-u-1$ zeros between them. Thus C' cannot be a subsequence of A_j. This is a contradiction. □

We now prove the approximation lower bounds for C-LCS$(k, 1)$. Suppose there is a polynomial-time algorithm that approximates C-LCS$(k, 1)$ within a factor $f(|\Sigma|) \cdot \hat{m}^{1/2-\epsilon}$ for some computable function f and some $\epsilon > 0$. Then we can obtain a polynomial-time algorithm that approximates Max-Clique (on a graph G of n vertices) within a factor $n^{1-\epsilon}$ as follows:

1. If $n < (2f(2))^{1/\epsilon}$, use a brute-force algorithm to find a maximum clique in G, then return the clique.
2. Construct a C-LCS$(k, 1)$ instance as in our reduction, use the $f(|\Sigma|) \cdot \hat{m}^{1/2-\epsilon}$-approximation algorithm to find a subsequence of length ℓ, then obtain a clique of size q in G following the reverse implication of Lemma 1. If $q \geqslant 1$, return the clique of size q. Otherwise, return any single vertex in G as a clique of size 1.

The algorithm clearly finds an optimal solution in constant time if it returns in step 1. Now assume that $n \geqslant (2f(2))^{1/\epsilon}$, and proceed to step 2. Let q^* be the maximum size of a clique in G. Let ℓ^* be the maximum length of a constrained common subsequence for the reduced C-LCS$(k, 1)$ instance. By Lemma 1, we have $\ell^* = (q^*+1)n - 1$. The algorithm finds a subsequence of length

$$\ell \geqslant \frac{\ell^*}{f(2) \cdot \hat{m}^{1/2-\epsilon}} = \frac{(q^*+1)n - 1}{f(2) \cdot \hat{m}^{1/2-\epsilon}} \geqslant \frac{q^*+1}{f(2) \cdot \hat{m}^{1/2-\epsilon}} n - 1,$$

then obtains a clique of size

$$q \geqslant \frac{q^*+1}{f(2) \cdot \hat{m}^{1/2-\epsilon}} - 1.$$

Recall that $\hat{m} = n^2$ and $n \geqslant (2f(2))^{1/\epsilon}$. It follows that

$$\max\{q, 1\} \geqslant \frac{q+1}{2} \geqslant \frac{q^*+1}{2f(2)\cdot \hat{m}^{1/2-\epsilon}} > \frac{q^*}{2f(2)\cdot n^{1-2\epsilon}} = \frac{n^\epsilon}{2f(2)} \cdot \frac{q^*}{n^{1-\epsilon}} \geqslant \frac{q^*}{n^{1-\epsilon}}.$$

Let us recall the following result of Zuckerman which improves an earlier result of Håstad [8]:

Theorem 6. *(See Zuckerman (2007) [15].) For any $\epsilon > 0$, there is no polynomial-time algorithm that approximates* Max-Clique *within a factor $n^{1-\epsilon}$ unless* NP = P.

By our reduction, it follows that for any computable function f and any $\epsilon > 0$, there is no polynomial-time algorithm that approximates C-LCS$(k, 1)$ within a factor $f(|\Sigma|)\cdot \hat{m}^{1/2-\epsilon}$ unless NP = P. The proof of Theorem 1 is now complete.

2.2. Proof of Theorem 3

It is easy to check that the reduction that Jiang and Li [10] used to prove the inapproximability of LCS over an arbitrary alphabet is an L-reduction from Max-Clique. Thus, in conjunction with the result of Zuckerman [15], this L-reduction actually implies the following theorem although it is not explicitly stated in their paper:

Theorem 7. *(See Jiang and Li (1995) [10].) For any $\epsilon > 0$, there is no polynomial-time algorithm that approximates* LCS *over an arbitrary alphabet within a factor $\hat{m}^{1-\epsilon}$ unless* NP = P*, where $\hat{m}$ is the length of the shortest input sequence.*

Since C-LCS$(k, 1)$ over an arbitrary alphabet includes LCS over an arbitrary alphabet as a special case, Theorem 3 immediately follows.

3. Improved approximation algorithms for C-LCS$(k, 1)$

In this section we prove Theorems 2 and 4.

3.1. Proof of Theorem 2

Let C^* be a constrained longest common subsequence. Since $B \preccurlyeq C^*$, we can embed B inside C^* in some fixed way such that $C^* = C_0 B[1] C_1 \ldots B[b] C_b$, then assign each index a between 0 and b (which we call a *slot*) a *value* that is the length of the subsequence C_a.

The previous approximation algorithm of Gotthilf et al. [6] is essentially a greedy algorithm that composes a constrained common subsequence from two parts: the constraint sequence B itself and a $|\Sigma|$-approximation of the subsequence C_a for a slot a of the highest value. Our improved algorithm for C-LCS$(k, 1)$ uses Gotthilf et al.'s greedy algorithm [6] as the first step, then supplements it with a brute-force algorithm and a random procedure. Instead of betting on a single large slot, the random procedure guesses a large number s of slots of high value. Intuitively, the random procedure and the greedy algorithm complement each other in their respective worst cases. Then an improved approximation ratio can be obtained by balancing them with suitably chosen parameters.

Algorithm A1.

1. Run Gotthilf et al.'s algorithm [6] to find a constrained common subsequence:
   (a) For each slot a, $0 \leqslant a \leqslant b$, do the following:
      (i) Partition the constraint sequence $B \to B[1, a]B[a+1, b]$.
      (ii) Partition each input sequence $A_i \to L_{i,a} M_{i,a} R_{i,a}$ such that $B[1, a] \preccurlyeq L_{i,a}$, $B[a+1, b] \preccurlyeq R_{i,a}$, and $M_{i,a}$ is maximal.
      (iii) Find a longest unary sequence M_a that is a common subsequence of $M_{i,a}$, $1 \leqslant i \leqslant k$.
   (b) Compose a sequence $B[1, a]M_a B[a+1, b]$ for a slot a such that $|M_a|$ is maximum.
2. Let z be the smallest positive integer[1] such that for all integers $\ell \geqslant z$,

$$\ell \geqslant 16\lceil \log \ell / \log\log \ell \rceil^3. \tag{1}$$

   For each integer ℓ, $b+1 \leqslant \ell < z$, use brute force to find a constrained common subsequence of length ℓ if it exists: enumerate all $|\Sigma|^\ell$ candidate sequences of length ℓ, and for each candidate sequence check whether it is a supersequence of the constraint sequence B and is a common subsequence of the input sequences A_i, $1 \leqslant i \leqslant k$.

[1] A calculation shows that $z = 324$. Our choice of this value is somewhat arbitrary. We set the parameter z to a concrete value here mostly for convenience, so that later in the analysis we can prove a concrete approximation ratio $\lambda \leqslant |\Sigma|\sqrt{\mathrm{OPT}\cdot \log\log \mathrm{OPT}/\log \mathrm{OPT}}$ without using the big-O notation. In actual implementation, we can set z to a smaller value, which results in a reduced running time of step 2 at the cost of an increased approximation ratio λ that is still $O(|\Sigma|\sqrt{\mathrm{OPT}\cdot \log\log \mathrm{OPT}/\log \mathrm{OPT}})$.

3. For each integer ℓ, $\max\{b+1, z\} \leqslant \ell \leqslant \hat{m}$, set the parameters

$$s = \lceil \log \ell / \log\log \ell \rceil, \qquad l = \left\lfloor (\ell \cdot \log\log \ell / \log \ell)^{1/2} \right\rfloor - 1, \quad \text{and} \quad w = \left\lceil \frac{l}{|\Sigma|} \right\rceil,$$

then for some tunable constant r (which controls the probability), repeat the following random procedure for $r(4s)^s$ rounds to find a constrained common subsequence of length $b + sw$:

(a) Randomly select (sample with replacement) s slots between 0 and b. Sort the s slots in ascending order: $0 \leqslant b_1 \leqslant \cdots \leqslant b_s \leqslant b$. Randomly select a letter $\sigma_i \in \Sigma$ for each slot b_i, $1 \leqslant i \leqslant s$.

(b) If the s slots are all distinct, that is, $0 \leqslant b_1 < \cdots < b_s \leqslant b$, compose a candidate constrained sequence

$$B[1, b_1]\sigma_1^w \ldots B[b_{s-1}+1, b_s]\sigma_s^w B[b_s+1, b],$$

and check whether it is a common subsequence of the input sequences A_i, $1 \leqslant i \leqslant k$.

4. Return the longest constrained sequence found.

Approximation ratio. Let OPT be the length of the constrained longest common subsequence C^*. Let APX_1, APX_2, and APX_3, respectively, be the maximum length of a constrained common subsequence found in step 1, step 2, and step 3 of the algorithm. Put $\lambda_1 = \mathrm{OPT}/\mathrm{APX}_1$, $\lambda_2 = \mathrm{OPT}/\mathrm{APX}_2$, $\lambda_3 = \mathrm{OPT}/\mathrm{APX}_3$, and $\lambda = \min\{\lambda_1, \lambda_2, \lambda_3\}$.

We clearly have $b \leqslant \mathrm{OPT} \leqslant \hat{m}$. If $\mathrm{OPT} = b$, then $\mathrm{APX}_1 = \mathrm{OPT}$ and $\lambda_1 = 1$. Also, if $b + 1 \leqslant \mathrm{OPT} < z$, then $\mathrm{APX}_2 = \mathrm{OPT}$ and $\lambda_2 = 1$. So suppose that

$$\max\{b+1, z\} \leqslant \mathrm{OPT} \leqslant \hat{m}.$$

Then OPT is equal to ℓ for some iteration in step 3.

Put

$$f = \sqrt{(\log \mathrm{OPT}/\log\log \mathrm{OPT})/|\Sigma|}. \tag{2}$$

We will show that

$$\lambda \leqslant \sqrt{\mathrm{OPT} \cdot |\Sigma|}/f,$$

hence

$$\lambda \leqslant |\Sigma| \sqrt{\mathrm{OPT} \cdot \log\log \mathrm{OPT}/\log \mathrm{OPT}} \leqslant |\Sigma| \sqrt{\hat{m} \cdot \log\log \hat{m}/\log \hat{m}}.$$

We first look at step 1. Let h be the highest value of a slot. Clearly,

$$h \geqslant \left\lceil \frac{\mathrm{OPT} - b}{b+1} \right\rceil \geqslant 1.$$

Gotthilf et al.'s algorithm [6] finds a constrained common subsequence of length

$$\mathrm{APX}_1 \geqslant b + \left\lceil \frac{h}{|\Sigma|} \right\rceil \geqslant \max\left\{ b+1, \frac{\mathrm{OPT}}{(b+1)|\Sigma|} \right\},$$

thus

$$\lambda_1 \leqslant \min\left\{ \frac{\mathrm{OPT}}{b+1}, (b+1)|\Sigma| \right\}.$$

If $h \geqslant \sqrt{\mathrm{OPT} \cdot |\Sigma|} \cdot f$, then $\mathrm{APX}_1 \geqslant \sqrt{\mathrm{OPT}/|\Sigma|} \cdot f$ and $\lambda_1 \leqslant \sqrt{\mathrm{OPT} \cdot |\Sigma|}/f$. So suppose that

$$h \leqslant \sqrt{\mathrm{OPT} \cdot |\Sigma|} \cdot f. \tag{3}$$

If $b + 1 \geqslant \sqrt{\mathrm{OPT}/|\Sigma|} \cdot f$ or $b + 1 \leqslant \sqrt{\mathrm{OPT}/|\Sigma|}/f$, then again $\lambda_1 \leqslant \sqrt{\mathrm{OPT} \cdot |\Sigma|}/f$. So suppose that

$$\sqrt{\mathrm{OPT}/|\Sigma|}/f \leqslant b + 1 \leqslant \sqrt{\mathrm{OPT}/|\Sigma|} \cdot f. \tag{4}$$

Now proceed to step 3. Consider the iteration where $\ell = \mathrm{OPT}$. From (2) we have

$$s = \lceil \log \mathrm{OPT}/\log\log \mathrm{OPT} \rceil = \left\lceil f^2 |\Sigma| \right\rceil, \tag{5}$$

and

$$l = \left\lfloor (\mathrm{OPT} \cdot \log\log \mathrm{OPT}/\log \mathrm{OPT})^{1/2} \right\rfloor - 1 = \left\lfloor \sqrt{\mathrm{OPT}/|\Sigma|}/f \right\rfloor - 1. \tag{6}$$

Also, from (1) we have

$$\mathrm{OPT} \geqslant 16s^3. \tag{7}$$

Let t be the number of slots of value at least l. Then the number of slots of value less than l is $b+1-t$. Since

$$\mathrm{OPT} \leqslant b + t \cdot h + (b+1-t) \cdot l,$$

we have

$$t \geqslant \frac{\mathrm{OPT} - b - (b+1)l}{h-l} \geqslant \frac{\mathrm{OPT} - (b+1)(l+1)}{h} \geqslant \frac{\mathrm{OPT} - \sqrt{\mathrm{OPT}/|\Sigma|} \cdot f \cdot \sqrt{\mathrm{OPT}/|\Sigma|}/f}{\sqrt{\mathrm{OPT} \cdot |\Sigma|} \cdot f}$$
$$= (1 - 1/|\Sigma|)\sqrt{\mathrm{OPT}/|\Sigma|}/f \geqslant \frac{1}{2}\sqrt{\mathrm{OPT}/|\Sigma|}/f, \tag{8}$$

where the third inequality follows from (3), (4), and (6). Then, from (8), (7), and (5) we have

$$t \geqslant \frac{1}{2}\sqrt{\frac{\mathrm{OPT}}{f^2|\Sigma|}} \geqslant \frac{1}{2}\sqrt{\frac{16s^3}{s}} = 2s, \tag{9}$$

and from (9) and (5) we have

$$b \geqslant \frac{b+1}{2} \geqslant \frac{t}{2} \geqslant s \geqslant 2f^2. \tag{10}$$

If a constrained common subsequence of length $b + sw$ is found in this iteration of step 3, then from (6) and (10) we have

$$\mathrm{APX}_3 \geqslant b + f^2|\Sigma| \cdot \frac{l}{|\Sigma|} \geqslant b + \sqrt{\mathrm{OPT}/|\Sigma|} \cdot f - 2f^2 \geqslant \sqrt{\mathrm{OPT}/|\Sigma|} \cdot f,$$

thus

$$\lambda_3 \leqslant \sqrt{\mathrm{OPT} \cdot |\Sigma|}/f.$$

Probability. We now estimate the probability that a constrained common subsequence is found in step 3 in the iteration where $\ell = \mathrm{OPT}$. First consider the probability p that a constrained common subsequence is found in one round of the random procedure. Since the random procedure always finds a constrained common subsequence if it guesses correctly s distinct slots of value at least l, and guesses correctly the dominating letter for each of the s slots, we have

$$p \geqslant \frac{t!/(t-s)!}{(b+1)^s} \cdot \frac{1}{|\Sigma|^s}. \tag{11}$$

From (9), we have $t!/(t-s)! \geqslant (t/2)^s$. From (8) and (4), we have $t/(b+1) \geqslant 1/(2f^2)$. Also recall (5) that $s = \lceil f^2|\Sigma| \rceil$. Thus

$$p \geqslant \left(\frac{t/2}{(b+1)|\Sigma|}\right)^s \geqslant \left(\frac{1}{4f^2|\Sigma|}\right)^s \geqslant \left(\frac{1}{4s}\right)^s.$$

Put $x = (4s)^s$. Then each round of the random procedure finds a constrained common subsequence with probability at least $1/x$. Since the random procedure is repeated for rx rounds, the probability that a constrained common subsequence is not found in rx consecutive rounds is at most

$$(1 - 1/x)^{rx} \leqslant 1/e^r,$$

which can be made arbitrarily small by choosing the constant r sufficiently large.

Time complexity. Steps 1 and 2 are clearly polynomial. For step 3 to be polynomial, it is sufficient that $s = O(\log n / \log\log n)$ so that $(4s)^s = \mathrm{poly}(n)$, where n is the input size. This is clearly satisfied since $s = \lceil \log \ell / \log\log \ell \rceil$ and $\ell \leqslant \hat{m} \leqslant n$.

3.2. Proof of Theorem 4

We obtain an $O(\hat{m}/\log\hat{m})$ approximation for C-LCS$(k, 1)$ over an arbitrary alphabet using Halldórsson's partitioning technique [7]. Assume without loss of generality that $\hat{m} \geqslant 2$.

Algorithm A2.

1. Find a shortest input sequence $\hat{A}$, which has length $\hat{m}$, and partition it into $q \leqslant \lceil \hat{m}/\log \hat{m} \rceil$ substrings $\hat{A} \to S_1 \dots S_q$ such that each substring S_p, $1 \leqslant p \leqslant q$, has length at most $\lceil \log \hat{m} \rceil$.
2. For each pair of indices u and v, $0 \leqslant u < v \leqslant b+1$, and for each subsequence T of each substring S_p, $1 \leqslant p \leqslant q$, compose a candidate constrained sequence

$$B[1,u]TB[v,b],$$

and check whether it is a supersequence of the constraint sequence B and is a common subsequence of the input sequences A_i, $1 \leqslant i \leqslant k$.
3. Return the longest constrained sequence found.

Approximation ratio. Let C^* be a constrained longest common subsequence. Since $\hat{A} = S_1 \dots S_q$ and $C^* \preccurlyeq \hat{A}$, we can partition C^* into q substrings $C^* \to C_1 \dots C_q$ such that $C_p \preccurlyeq S_p$ for $1 \leqslant p \leqslant q$. Similarly, since $C^* = C_1 \dots C_q$ and $B \preccurlyeq C^*$, we can partition B into q substrings $B \to T_1 \dots T_q$ such that $T_p \preccurlyeq C_p$ for $1 \leqslant p \leqslant q$.

By the Pigeonhole principle, at least one of the q substrings of C^*, say C_p, has length at least $1/q$ times the length of C^*. This substring C_p is enumerated by the algorithm as some subsequence T of S_p. Write $T_1 \dots T_{p-1} = B[1,u]$ and $T_{p+1} \dots T_q = B[v,b]$. Then

$$B = T_1 \dots T_{p-1} T_p T_{p+1} \dots T_q \preccurlyeq T_1 \dots T_{p-1} C_p T_{p+1} \dots T_q = B[1,u]TB[v,b]$$

and

$$B[1,u]TB[v,b] = T_1 \dots T_{p-1} C_p T_{p+1} \dots T_q \preccurlyeq C_1 \dots C_{p-1} C_p C_{p+1} \dots C_q = C^*.$$

The length of $B[1,u]TB[v,b]$ is at least the length of T, which is at least $1/q \geqslant 1/\lceil \hat{m}/\log \hat{m} \rceil$ times the length of C^*.

Time complexity. The dominating step of the algorithm is step 2. There are $O(b^2)$ pairs of indices u and v, $\lceil \hat{m}/\log \hat{m} \rceil$ substrings S_p, and at most $2^{\lceil \log \hat{m} \rceil} = O(\hat{m})$ subsequences T of each substring S_p. Thus the total number of candidate constrained sequences is $O(b^2 \hat{m}^2/\log \hat{m})$. For each candidate constrained sequence, it takes $O(n)$ time to check whether it is valid. The overall running time of the algorithm is polynomial.

4. Exact algorithms for C-LCS(k, l)

In this section we prove Theorem 5 by presenting two exact algorithms for C-LCS(k,l).

Our first exact algorithm, which runs in $O(|\Sigma|^{\mathrm{OPT}+1} \cdot n)$ time, is a trivial brute-force algorithm: for $\ell = 1, \dots, \hat{m}$, enumerate all $|\Sigma|^\ell$ sequences of length ℓ, then for each candidate sequence check in $O(n)$ time whether it is a constrained common subsequence; stop the iteration if for some ℓ no candidate sequence of length ℓ is a constrained common subsequence.

Our second exact algorithm is based on dynamic programming, and achieves a running time of $O(\prod_{i=1}^{k}(|A_i|+1) \cdot \prod_{j=1}^{l}(|B_j|+1) \cdot (k+l))$. For simplicity, we only compute the maximum length of a constrained common subsequence (or report that the problem has no solution). By standard techniques, an actual constrained common subsequence of the maximum length can be found (if it exists) within the same running time.

Denote by $L(a_1, \dots, a_k; b_1, \dots, b_l)$ the maximum length of a constrained common subsequence for the subproblem with partial input sequences $A_1[1,a_1], \dots, A_k[1,a_k]$ and partial constraint sequences $B_1[1,b_1], \dots, B_l[1,b_l]$, where $0 \leqslant a_i \leqslant |A_i|$ and $0 \leqslant b_j \leqslant |B_j|$ for $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant l$. We use the value $-\infty$ to indicate that a subproblem has no solution. The desired entry is $L(|A_1|, \dots, |A_k|; |B_1|, \dots, |B_l|)$.

The base cases are

$$L(0, \dots, 0; 0, \dots, 0) = 0,$$

and

$$L(a_1, \dots, a_k; b_1, \dots, b_l) = -\infty, \quad \text{if } \min_{1 \leqslant i \leqslant k} a_i < \max_{1 \leqslant j \leqslant l} b_j.$$

The recurrence is

$$L(a_1, \dots, a_k; b_1, \dots, b_l) = \max \begin{cases} \max_{1 \leqslant i \leqslant k} L(a_1, \dots, a_{i-1}, a_i - 1, a_{i+1}, \dots, a_k; b_1, \dots, b_l), \\ L(a_1 - 1, \dots, a_k - 1; b'_1, \dots, b'_l) + 1, \end{cases}$$

where the second case applies only if $A_1[a_1] = \cdots = A_k[a_k] = \sigma$ for some $\sigma \in \Sigma$; then we let $b'_j = b_j - 1$ if $B_j[b_j] = \sigma$ and let $b'_j = b_j$ if $B_j[b_j] \neq \sigma$.

The table L has $\prod_{i=1}^{k}(|A_i|+1) \cdot \prod_{j=1}^{l}(|B_j|+1)$ entries in total; each entry can be computed in $O(k+l)$ time. Hence the overall running time of the dynamic programming algorithm is $O(\prod_{i=1}^{k}(|A_i|+1) \cdot \prod_{j=1}^{l}(|B_j|+1) \cdot (k+l))$.

References

[1] H.-Y. Ann, C.-B. Yang, C.-T. Tseng, C.-Y. Hor, Fast algorithms for computing the constrained LCS of run-length encoded strings, in: Proceedings of the 2009 International Conference on Bioinformatics & Computational Biology (BIOCOMP'09), 2009, pp. 646–649.
[2] A. Arslan, Ö. Eğecioğlu, Algorithms for the constrained longest common subsequence problems, Internat. J. Found. Comput. Sci. 16 (2005) 1099–1109.
[3] P. Bonizzoni, G. Della Vedova, R. Dondi, Y. Pirola, Variants of constrained longest common subsequence, Inform. Process. Lett. 110 (2010) 877–881.
[4] Y.-C. Chen, K.-M. Chao, On the generalized constrained longest common subsequence problems, J. Comb. Optim. 21 (2011) 383–392.
[5] F.Y.L. Chin, A. De Santis, A.L. Ferrara, N.L. Ho, S.K. Kim, A simple algorithm for the constrained sequence problems, Inform. Process. Lett. 90 (2004) 175–179.
[6] Z. Gotthilf, D. Hermelin, M. Lewenstein, Constrained LCS: hardness and approximation, in: Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM'08), 2008, pp. 255–262.
[7] M.M. Halldórsson, Approximation via partitioning. Technical Report IS-RR-95-0003F, School of Information Science, Japan Advanced Institute of Science and Technology, Hokuriku, 1995.
[8] J. Håstad, Clique is hard to approximate within $n^{1-\epsilon}$, Acta Math. 182 (1999) 105–142.
[9] C.S. Iliopoulos, M.S. Rahman, New efficient algorithms for the LCS and constrained LCS problems, Inform. Process. Lett. 106 (2008) 13–18.
[10] T. Jiang, M. Li, On the approximation of shortest common supersequences and longest common subsequences, SIAM J. Comput. 24 (1995) 1122–1139.
[11] D. Maier, The complexity of some problems on subsequences and supersequences, J. ACM 25 (1978) 322–336.
[12] M. Middendorf, On finding minimal, maximal, and consistent sequences over a binary alphabet, Theoret. Comput. Sci. 145 (1995) 317–327.
[13] M. Middendorf, D.F. Manlove, Combined super-/substring and super-/subsequence problems, Theoret. Comput. Sci. 320 (2004) 247–267.
[14] Y.-T. Tsai, The constrained longest common subsequence problem, Inform. Process. Lett. 88 (2003) 173–176.
[15] D. Zuckerman, Linear degree extractors and the inapproximability of max clique and chromatic number, Theory Comput. 3 (2007) 103–128.