Longest Common Subsequence Problem for Unoriented and Cyclic Strings

François Nicolas, Eric Rivals

Longest Common Subsequence Problem for Unoriented and Cyclic Strings

François Nicolas and Eric Rivals

L.I.R.M.M., CNRS U.M.R. 5506
161 rue Ada, F-34392 Montpellier Cedex 5, France
{nicolas, rivals}@lirmm.fr

Abstract. Given a finite set of strings X, the longest common subsequence problem (LCS) consists in finding a longest common subsequence of the strings in X. LCS is a central problem in stringology and finds broad applications in string correction, text compression, conception of error-detecting codes, or biological sequence comparison. However, in numerous contexts words represent cyclic or unoriented sequences of symbols and LCS must be generalized to consider both orientations and/or all cyclic shifts of the strings involved. This occurs especially in computational biology where genetic material is sequenced from circular DNA or RNA molecules. In this work, we define three variants of LCS where the input words are unoriented and/or cyclic. We show that these problems are NP-hard even when restricted to instances over a binary alphabet, and W[1]-hard if parameterized in the number of input strings or in the length of the sought subsequence. We also study the approximability of these variants and conclude that they are as hard to approximate as Maximum Independent Set. We obtain this through a reduction from Maximum Independent Set in Hypergraphs for which we also prove similar results.

1 Introduction

1.1 The Longest Common Subsequence problem

Let s, t be two words and denote the length of a word s by $|s|$. s is a *subsequence* (or a subword) of t if s can be obtained by erasing zero or more symbols of t. In that case, t is a *supersequence* of s. For a set of words X, finding a word s that is a subsequence common to all words in X and such that $|s|$ is maximal is known as the Longest Common Subsequence problem (LCS). The dual problem of finding a Shortest Common Supersequence for the input X is denoted SCS. The LCS problem, as well as the SCS problem, find numerous applications, for instance in string correction, text compression, conception of error-detecting codes, biological sequence comparison, and many more (see [21] for a review). Moreover, it is of theoretical interest as a central problem in stringology and as a simpler version of the more general Multiple Alignment problem [12, Chapter 14].

1.2 Examples of cyclic and/or unoriented words

We review some domains in which linear words represent cyclic sequences of symbols.

In nature, inherited information is stored on linear or circular DNA or RNA molecules [2]. Bacterial genomes are in majority circular, as well as chloroplasts and mitochondrial genomes. Viruses often store their genetic material on circular DNA, especially bacteriophages, which are widely used as vectors for cloning a gene of another species. Plasmids, small circular DNA molecules that have the ability to replicate on their own, are extensively used in biotechnology [2]. All such cyclic molecules are sequenced and represented as linear strings by choosing an arbitrary starting point. It follows that the comparison of two such sequences needs to consider all possible cyclic shifts (also called *conjugates*) of one of the sequences. This is cyclic string comparison.

DNA, as well as RNA, are oriented molecules. In the sequencing process that is now carried out automatically, the DNA of interest is cloned into a double-stranded vector and the vector is sequenced. For technical reasons, it is not always possible to know in which orientation the DNA is inserted into the vector (i.e., on which strand). Therefore, in some cases, e.g., for Expressed Sequence Tags, sequences must be compared in each orientation [1]. We call this unoriented comparison.

Another domain in which cyclic strings arise is pattern representation and recognition [7]. There, the closed contour of a two-dimensional (polygonal) shape is encoded into a linear string by choosing arbitrarily a start position on the contour. Determining if two shapes are similar or equal requires to compare one string with all cyclic shifts of the other. Practically, this type of comparison is applied, for instance in an industrial context, to recognize the class of an object currently visible on a conveyor belt.

1.3 Known results

A large literature is devoted to LCS; we summarize below the main results.

The case of two input strings. Suppose given two words s and t with $|s| \leq |t|$. The first algorithm shows that a longest common subsequence (lcs) of s and t can be computed in $O(|s|\,|t|)$ time and $O(|s|)$ space by dynamic programming ([21, Chap. 12]). This can be improved to $O(|s|\,|t| \,/\, \log|t|)$ time [19]. Other algorithms yield better complexities when the problem is parameterized, e.g., by the number of identities between the two words (see [3, Chap. 4] for a review). The complexity of $O(|s|\,|t| \,/\, \log|t|)$ can also be achieved to compute an optimum alignment between s and t with unrestricted cost matrices [9].

The problem of finding a lcs between any cyclic shift of s and any cyclic shift of t has also been studied. Little thinking shows that this is equivalent to finding the lcs between s and any cyclic shift of t. Application of dynamic programming to compute the lcs of all combinations yields a brute-force algorithm in $O\left(|s|^2\,|t|\right)$ time. Schmidt [22] gives the first algorithm in $O(|s|\,|t|)$ time, whereas Maes exhibits an algorithm that computes an optimum alignment between any cyclic shift of s and any cyclic shift of t with unrestricted cost matrices in $O(|s|\,|t|\log|s|)$ time [17].

The case of many input strings. When the number of input strings is unbounded, LCS becomes intractable: it is proven to be NP-hard even when the alphabet is binary in [18], W[1]-hard when parameterized by the number of input strings in [20], and inapproximable for unbounded alphabets in [15]. The parameterized complexity with respect to (w.r.t.) several other parameters is also studied in [4, 5], while the average error of approximation algorithms is also investigated in [15]. These questions remain open for the variants of LCS where the input strings are cyclic and/or unoriented.

1.4 Our contribution

In this work, we define three variants of LCS when considering a set of input words that are i/ unoriented, ii/ cyclic or iii/ both. We show that these problems are first, NP-hard even when restricted to instances over a binary alphabet, and second W[1]-hard if parameterized by the number of input strings or in the length of the sought subsequence. We study the approximability of these variants and conclude that they are as hard to approximate as Maximum Independent Set. We obtain this through a reduction of Maximum Independent Set in Hypergraphs (MISH), for which we also prove similar results.

The paper is organized as follows. Section 2 gives basic notations, defines the problems and summarizes useful known results on the MISH problems. Section 3 studies the classical and parameterized complexity of our three variants of LCS, while Section 4 tackles with their approximability. We conclude and list open questions in the last section.

2 Notations and Definitions

2.1 Approximability

Let Max be a *maximization* problem. To any *instance* x of Max is associated a set $S(x)$ of *solutions*. Let (x, s) be a pair where x is an instance of Max and $s \in S(x)$ a solution. To each pair (x, s) is associated a *measure* $\mu(x, s) \in \mathbb{N}$ of the quality of solution s for the instance x. The measure for an optimal solution is $\max_{s \in S(x)} \mu(x, s)$

Let ρ be a function that maps an instance of Max to a real number larger than or equal to 1. An *approximation algorithm with bound* ρ for Max is an algorithm that for each input instance x of Max returns a solution $a \in S(x)$ satisfying $\rho(x)\mu(x, a) \geq \max_{s \in S(x)} \mu(x, s)$.

The *decision problem associated with* Max, denoted by Max_D, can be stated as follows.

Problem (Max_D). *Let (x, k) be a pair where x is an instance of* Max *and k a positive integer. Does it exist $s \in S(x)$ satisfying $\mu(x, s) \geq k$?*

2.2 Words, Subsequences and Languages

Words and Languages. We denote the cardinal of any finite set A by $\#A$. An *alphabet* Σ is a finite set of *letters*. A *word* over Σ is a finite sequence of elements of Σ. The set of all words over Σ is denoted by $\Sigma^\star$. The empty sequence, denoted by ε, is called the *empty word*. For a word x, $|x|$ denotes the length of x. Given two words x and y, we denote by xy the *concatenation* of x and y. For all $i \in [1, |x|]$, $x[i]$ denotes the i-th letter of x: $x = x[1]x[2] \ldots x[|x|]$. For all letter a, $|x|_a := \#\{i \in [1, |x|] : x[i] = a\}$ denotes the number of *occurrences* of the letter a in x. For all $n \in \mathbb{N}$, we denote by x^n the *n-th power* of x that is, the concatenation of n copies of x (note that $x^0 = \varepsilon$). A word x is *unary* if it exits a letter a such that $x = a^{|x|}$. The *mirror image* of x is the word $\tilde{x} := x[|x|] \cdots x[2]x[1]$. Two words x and y are *conjugates* of each other if there exists two words u and v such that $x = uv$ and $y = vu$. A *language* over Σ is any set of words over Σ. We denote by $\sigma(X)$ the cardinal of the smallest alphabet Σ such that $X \subseteq \Sigma^\star$.

Subsequences of a Word. In addition to the usual notion of subword or subsequence, we define three more types of subsequences.

Definition 1. *Let s and x be two words. We say that:*

1. *s is a* subword *of x if one obtains s by erasing some letters of x (eventually all or none),*
2. *s is an* unoriented subword *(*U*-subword) of x when s or $\tilde{s}$ is a subword of x,*
3. *s is a* cyclic subword *(*C*-subword) of x when a conjugate of s is a subword of x,*
4. *s is an* unoriented cyclic subword *(*UC*-subword) of x when a conjugate of s is a* U*-subword of x.*

Remark 1. For any words x and s one has:

– s is a U-subword of x iff s is a subword of x or of $\tilde{x}$,
– s is a C-subword of x iff s is a subword of a conjugate of x,
– s is a UC-subword of x iff s is a U-subword of a conjugate of x,
– any subword of x is both a U-subword and a C-subword of x,
– any U-subword of x and any C-subword of x are UC-subwords of x.

Longest Common Subword of a Language. Let X be a non empty language. A *common* subword (resp. U-subword, resp. C-subword, resp. UC-subword) of X is a word that is subword (resp. U-subword, resp. C-subword, resp. UC-subword) of each word in X.

Definition 2. *For any non empty language X, we denote by* $\mathrm{lcs}(X)$ *(resp.* $\mathrm{lcus}(X)$*, resp.* $\mathrm{lccs}(X)$*, resp.* $\mathrm{lcucs}(X)$*) the length of a longest common subword (resp.* U*-subword, resp.* C*-subword, resp.* UC*-subword) of X.*

Remark 2. For any non empty language X, it follows from Remark 1 that

$$\mathrm{lcs}(X) \leq \mathrm{lcus}(X) \leq \mathrm{lcucs}(X) \quad \text{and} \quad \mathrm{lcs}(X) \leq \mathrm{lccs}(X) \leq \mathrm{lcucs}(X)\,.$$

Example 1. One easily checks that in the following example that each of these inequalities may be strict, which guarantees the consistency of Definition 2.

– $\mathrm{lcs}(X) = 1 < 2 = \mathrm{lcus}(X) = \mathrm{lcucs}(X)$ for $X := \{\mathtt{01}, \mathtt{10}\}$,
– $\mathrm{lcus}(Y) = 2 < 3 = \mathrm{lccs}(Y) = \mathrm{lcucs}(Y)$ for $Y := \{\mathtt{011}, \mathtt{101}\}$ and
– $\mathrm{lccs}(Z) = 5 < 6 = \mathrm{lcus}(Z) = \mathrm{lcucs}(Z)$ for $Z := \{\mathtt{101001}, \mathtt{100101}\}$.

Definitions of the Problems. To each of these notions of common subword corresponds an optimization problem. We call respectively

– Longest Common Subsequence (LCS),
– Longest Common Unoriented Subsequence (LCUS),
– Longest Common Cyclic Subsequence (LCCS) and
– Longest Common Unoriented Cyclic Subsequence (LCUCS)

the following maximization problems:

Problem (LCS (resp. LCUS, resp. LCCS, resp. LCUCS)). *Let X be a non empty finite language, find a common subword (resp.* U*-subword, resp.* C*-subword, resp.* UC*-subword) of X of maximal length.*

2.3 Independent Sets in Graphs and Hypergraphs

Definitions. A *hypergraph* is a pair $H = (\mathrm{V}(H), \mathrm{E}(H))$ where $\mathrm{V}(H)$ is a finite set and $\mathrm{E}(H)$ is a set of subsets of $\mathrm{V}(H)$. The elements of $\mathrm{V}(H)$ are the *vertices* of H and the ones of $\mathrm{E}(H)$ are the *hyperedges* of H. We denote by $|H| := \#\mathrm{V}(H)$ the number of vertices in H. A hypergraph *on* V (where V is any finite set) is a hypergraph H such that $\mathrm{V}(H) = V$.

An *independent set* of H is a set of vertices of H that contains no hyperedge of H. We denote by $\alpha(H)$ the maximum cardinality of an independent set of H. One calls Maximum Independent Set in Hypergraphs (MISH) the following optimization problem:

Problem (MISH). *Let H be a hypergraph, find an independent set of H whose cardinal is maximal.*

A r*-uniform* hypergraph (where $r \in \mathbb{N} \setminus \{0, 1\}$) is a hypergraph whose hyperedges have cardinal r. A *graph* is a 2-uniform hypergraph. The hyperedges of a graph are called *edges*. For any $r \in \mathbb{N} \setminus \{0, 1\}$, we denote by r-MISH the restriction of MISH to r-uniform hypergraphs, H. The instances of 2-MISH, which are graphs, are denoted by G instead of H.

Known Results. The decision problem associated with 2-MISH is NP-complete [8] and W[1]-complete for parameter k [10]. The maximization problem MISH admits a polynomial approximation algorithm with bound $O(|H| / \log |H|)$ [13]. In the case of 2-MISH, there exists a polynomial approximation algorithm with bound $O\big(|G| / \log^2 |G|\big)$ [6]. For MISH, the trivial algorithm that for each input hypergraph H returns an independent set with cardinal 1 yields bound $|H|$, which is only slightly worse than the previous one.

Further studies on the approximability of 2-MISH show that the existence of a polynomial approximation algorithm with bound $|G|^{\delta(|G|)}$ where $\delta : \mathbb{N} \to [0,1]$ implies the unprobable inclusion of NP in various complexity classes according to the asymptotic behavior of δ:

- if $\sup_{n\in\mathbb{N}} \delta(n) < \frac{1}{2}$ then NP = P [14],
- if $\sup_{n\in\mathbb{N}} \delta(n) < 1$ then NP = ZPP [14] and
- if $\delta(n) = 1 - O\big((\log\log n)^{-1/2}\big)$ then NP $\subseteq$ ZPTIME $\Big(2^{O\left((\log n)(\log\log n)^{3/2}\right)}\Big)$. [11]

3 Hardness of LCUS, LCCS, and LCUCS

In this section, we show that the decision problems associated with LCUS, LCCS, and LCUCS are NP-complete and W[1]-hard when parameterized by the number of input words. Our proofs are valid also in the case of *binary* input languages.

Let us first demonstrate a synchronization lemma.

Lemma 1. *Let a, b be two distinct letters, and s, t be two words. Let m, n, p, q be four strictly positive integers satisfying $|s| < \min\{p, q\}$. Set $u := a^m s b^n$ and $v := a^p t b^q$. Then, v is a conjugate of u or of its mirror image iff $u = v$.*

Proof. Any conjugate of u that is *distinct from* u adopts one of the three following forms:

(*i*) $a^k s b^n a^{m-k}$ with $k \in [1, m-1]$,
(*ii*) $b^k a^m s b^{n-k}$ with $k \in [1, n-1]$,
(*iii*) $s_2 b^n a^m s_1$ where s_1 and s_2 are two words such that $s = s_1 s_2$.

• Assume that v is a conjugate of u. The word v ends by letter b, so it is not of the form (*i*), and v begins with letter a, so it is not of the form (*ii*). Moreover, as $|s|$ is strictly lower than p, v admits $a^{|s|+1}$ as a prefix, which is not the case of words of the form (*iii*). Thus, v is a conjugate of u which is not of any of these three forms. The only possibility left is $u = v$.

• Let us suppose that $\tilde{v}$ is a conjugate of u distinct from u. As $\tilde{v} = b^q \tilde{t} a^p$ begins by letter b, it does not adopt form (*i*), and as $\tilde{v}$ ends by letter a, it does not adopt form (*ii*) either. It follows that $\tilde{v}$ has form (*iii*) and thus, there exists two words s_1, s_2 such that $s = s_1 s_2$ and $b^q \tilde{t} a^p = s_2 b^n a^m s_1$. Since by hypothesis $q \geq |s| \geq |s_2|$, one has $s_2 = b^{|s_2|}$ and similarly, $p \geq |s| \geq |s_1|$ requires $s_1 = a^{|s_1|}$. We obtain that $\tilde{v} = b^{|s_2|+n} a^{|s_1|+m}$ and $u = a^m s_1 s_2 b^n = a^{m+|s_1|} b^{n+|s_2|}$, which yields $u = v$, a contradiction. □

For any finite language X and any distinct letters a, b, we define:

$$m_X := \max_{x\in X} |x| \quad \text{and} \quad T_X^{a,b} := \left\{ a^{2m_X+1} x b^{2m_X+1} : x \in X \right\}$$

Thanks to the synchronization lemma, we obtain the following property.

Lemma 2.

$$\operatorname{lcucs}(T_X^{a,b}) = \operatorname{lccs}(T_X^{a,b}) = \operatorname{lcus}(T_X^{a,b}) = \operatorname{lcs}(T_X^{a,b}) = \operatorname{lcs}(X) + 4m_X + 2\,.$$

Proof. One checks easily that, for any non empty language W and any letter c, $\mathrm{lcs}(Wc) = \mathrm{lcs}(W) + 1$, where Wc means the language formed by concatenation of letter c to each word in W. Repeatedly applying this property, we get $\mathrm{lcs}(T_X^{a,b}) = \mathrm{lcs}(X) + 4m_X + 2$. It remains to show that $\mathrm{lcucs}(T_X^{a,b}) = \mathrm{lccs}(T_X^{a,b}) = \mathrm{lcus}(T_X^{a,b}) = \mathrm{lcs}(T_X^{a,b})$ or equivalently, according to Remark 2, that $\mathrm{lcucs}(T_X^{a,b}) \leq \mathrm{lcs}(T_X^{a,b})$.

Let v be a common UC-subword of $T_X^{a,b}$ of length $\mathrm{lcucs}(T_X^{a,b})$. v has maximal length. For any $x \in X$, there exists a subword u_x of $a^{2m_X+1}xb^{2m_X+1}$ that is a conjugate of v or of $\tilde{v}$. Consider the family of words $(u_x)_{x \in X}$; for all x, $y \in X$, we have

- $|u_x| = \mathrm{lcucs}(T_X^{a,b})$ because u_x has the same length as v,
- u_x is a subword of $a^{2m_X+1}xb^{2m_X+1}$ by definition,
- u_x is a conjugate of u_y or $\tilde{u}_y$, by the transitivity of conjugacy.

First, let us choose x, $y \in X$ and demonstrate that $u_x = u_y$. Words u_x and u_y can be written as $u_x = a^m s b^n$ and $u_y = a^p t b^q$ with s subword of x, t subword of y, and m, n, p, $q \in [0, 2m_X + 1]$. By contradiction, assume $p \leq m_X$; then we would have

$$\begin{aligned} \mathrm{lcucs}(T_X^{a,b}) = |u_y| = p + |t| + q &\leq m_X + |y| + 2m_X + 1 \\ &\leq 4m_X + 1 \\ &< 4m_X + 2 \, = \mathrm{lcs}(T_X^{a,b})\,. \end{aligned}$$

which, by Remark 2, is a contradiction.
We know now that $p > m_X \geq |x| \geq |s|$ and m, n, $q > |s|$ for the same reasons. Thus, Lemma 1 with $(u, v) := (u_x, u_y)$ can be applied and we get $u_x = u_y$.

The u_x ($x \in X$) are all equal to the same word u, which is a common subword of $T_X^{a,b}$. Therefore, $\mathrm{lcs}(T_X^{a,b}) \geq |u| = \mathrm{lcucs}(T_X^{a,b})$, what we wanted. □

We can now state the hardness of our problems in the following theorem.

Theorem 1. *The problems* LCUCS_D*,* LCCS_D *and* LCUS_D *are*

- NP*-complete even if restricted to binary instances, i.e., instances such that* $\sigma(X) = 2$*,*
- W[1]*-hard for parameter* $\#X$ *even when restricted to binary instances,*
- W[t]*-hard for parameter* $(\#X, \sigma(X))$ *for any* $t \in \mathbb{N} \setminus \{0\}$*.*

Proof. Let X be a non-empty finite language and let $k \in \mathbb{N}$. If X is unary, then the problem are polynomial since $\mathrm{lcs}(X) = \mathrm{lcus}(X) = \mathrm{lccs}(X) = \mathrm{lcucs}(X) = \min_{x \in X} |x|$. Now, let us that $\sigma(X) > 1$, i.e., at least two distinct letters occurs in X. Consider a polynomial function that to each non-empty finite language X associates a pair of distinct letters, (a_X, b_X) occurring in a word of X. By Lemma 2, the function

$$(X, k) \longmapsto \begin{cases} (T_X^{a_X, b_X}, k + 4m_X + 2) & \text{if } \sigma(X) \geq 2 \\ (X, k) & \text{otherwise} \end{cases}$$

is a valid reduction of LCS_D to LCUS_D, LCCS_D and LCUCS_D. Moreover, our reduction is computable in polynomial time and preserves the parameters $\#X$ and $\sigma(X)$, since $\#T_X^{a_X, b_X} = \#X$ and $\sigma(T_X^{a_X, b_X}) = \sigma(X)$.

This allows us to generalize to LCUS_D, LCCS_D and LCUCS_D the intractability results established so far for LCS_D: NP-completeness even when $\sigma(X) = 2$ in [18], W[1]-hardness for parameter $\#X$ even when $\sigma(X) = 2$ in [20], and W[t]-hardness for parameter $(\#X, \sigma(X))$ for any $t \in \mathbb{N} \setminus \{0\}$ in [4]. □

4 Approximability

This section is divided in two. In Section 4.1, we generalize to r-uniforms hypergraphs approximability results obtained on the Maximum Independent Set problem in graphs (2-MISH). In Section 4.2, we exhibit reductions of 2-MISH and 3-MISH to the unoriented and cyclic variants of LCS, which enables us to conclude on the approximability of the latter.

4.1 Approximability of r-MISH

The following theorem states that for any $r \in \mathbb{N} \setminus \{0,1\}$, r-MISH is at least as difficult as 2-MISH from the view-points of approximation and parameterized complexity.

Lemma 3. *Given $p \in \mathbb{N}$, it exists a mapping computable in polynomial time that maps any hypergraph H to an independent set of H, denote it J_G^p, having cardinal $\min\{\alpha(H), p\}$.*

Proof. For fixed p, enumerating all $O(|H|^p)$ p-subsets of $\mathrm{V}(H)$ and selecting those that are independent sets of H takes polynomial time. □

Theorem 2. *Let $r \in \mathbb{N} \setminus \{0,1\}$ and $b : \mathbb{N} \to [1, \infty[$.*

- *The decision problem r-MISH$_D$ is W[1]-hard for parameter k.*
- *If it exists a polynomial approximation algorithm for r-MISH with bound $b(|H|)$ then it also exists a polynomial approximation algorithm for 2-MISH with bound $b(|G|)$.*

Proof. For any graph G, consider the set of r-subsets of $\mathrm{V}(G)$ that contains at least an edge of G. We denote by H_G the r-uniform hypergraph on $\mathrm{V}(G)$ whose set of hyperedges is this abovementioned set. The proof relies on the three following properties of H_G:

P1. Any independent set of G is also an independent set of H_G, since any edge of H_G contains an edge of G.
P2. Any independent set of H_G whose cardinal is at least r is an independent set of G, for any r-subset of $\mathrm{V}(G)$ that contains an edge of G encloses also an edge of H.
P3. The mapping that associates to a graph G the hypergraph H_G is computable in polynomial time. Indeed, for a given r, one can enumerate the $\binom{|G|}{r} = O(|G|^r)$ r-subsets of $\mathrm{V}(G)$ and select those that contain an edge of G in polynomial time.

Note that H_G and G do not have the same independent sets.

• Let us now demonstrate that r-MISH$_D$ is W[1]-hard for parameter k. Let G be a graph and $k \in \mathbb{N}$. Consider the mapping

$$(G,k) \longmapsto \begin{cases} (H_G, k) & \text{if } k \geq r \\ ((J_G^r, \emptyset), k) & \text{if } k < r \end{cases}$$

It associates G to H_G if $k \geq r$, in which case G and H_G share any independent set whose cardinal is larger than r (consequence of Properties P1 and P2). Otherwise it associates G to $(J_G^r, \emptyset)$, i.e., an independent set of G of maximal cardinality. The previous remark implies that our mapping is a valid reduction of 2-MISH$_D$ to r-MISH$_D$. Moreover, as a consequence of P3 and of Lemma 3, it is computable in polynomial time. As our reduction preserves parameter k, the W[1]-hardness of r-MISH$_D$ for parameter k follows from the W[1]-hardness of 2-MISH$_D$ [10].

• Let us prove the announced approximation bound. Assume it exists a polynomial approximation algorithm with bound $b(|H|)$ for r-MISH. We describe a polynomial approximation algorithm with bound $b(|G|)$ for 2-MISH. For any input graph G, the algorithm proceeds as follow.

1. Compute the r-uniform hypergraph H_G,
2. Compute I, an independent set of H_G whose cardinal is at least $\alpha(H_G)/b(|H_G|)$,
3. If the cardinal of I is at least r, then return I,
4. Otherwise, return J_G^r.

First, the four steps described above can be performed in polynomial time: Step 1 because of P3, Step 2 by hypothesis, and Step 4 because of Lemma 3. It remains to show that the algorithm yields the right bound, i.e., $b(|G|)$.

We have $|G| = |H_G|$ since $\mathrm{V}(G) = \mathrm{V}(H_G)$, and $\alpha(G) \leq \alpha(H_G)$ because of P1. Thus, the independent set I computed during Step 2 has at least $\alpha(G)/b(|G|)$ vertices. When $\#I \geq r$, P2 guarantees that the set returned by the algorithm, I, is an independent set of G. In the other case, when $\#I < r$, the algorithm returns J_G^r, an independent set of G, such that

$$\#J_G^r = \min\{\alpha(G), r\} \geq \min\left\{\frac{\alpha(G)}{b(|G|)}, \#I\right\} = \frac{\alpha(G)}{b(|G|)}\,.$$

In both cases, our algorithm outputs an independent set of G whose cardinal is at least $\alpha(G)/b(|G|)$, which is the right bound. This concludes the proof. □

Note that [16] uses the transformation $G \mapsto H_G$ described in the preceding proof to show a similar result about the Graph Coloring problem.

4.2 Approximability of LCS, LCUS, LCCS and LCUCS

The following proposition is one of the few approximation algorithm for LCS and its variants. Note that the bound $\sigma(X)$ is announced for LCS in [15].

Proposition 1. *Each problem among* LCS*,* LCUS*,* LCCS *and* LCUCS *admits a polynomial approximation algorithm with bound* $\min\{\sigma(X), l/\log l\}$ *where* $l := \min_{x\in X} |x|$*.*

Proof. We proceed in two steps, one for each approximation bound.

• Existence of a polynomial approximation algorithm with bound $\sigma(X)$.

Consider the algorithm that maps any non empty finite language X to w_X, one maximal common *unary* subword of X. It chooses a letter a that occurs at least in a word of X and that maximizes the quantity $n_a := \min_{x\in X} |x|_a$. It returns $w_X := a^{n_a}$. All this requires polynomial time.

We now show that our algorithm has bound $\sigma(X)$, i.e., that u_X has length at least $\mathrm{lcs}(X)/\sigma(X)$ (resp. $\mathrm{lcus}(X)/\sigma(X)$, resp. $\mathrm{lccs}(X)/\sigma(X)$, resp. $\mathrm{lcucs}(X)/\sigma(X)$). By Remark 2, it suffices to prove that $|u_X| \geq \mathrm{lcucs}(X)/\sigma(X)$. The hint is that in any word s (it contains $\sigma(\{s\})$ distinct letters), it exists a letter whose number of occurrences in s is at least $|s|/\sigma(\{s\})$. Now, let s be a common UC-subword of X of maximal length $\mathrm{lcucs}(X)$. The preceding observation implies the existence of a letter, say b, such that $|s|_b \geq |s|/\sigma(\{s\}) \geq |s|/\sigma(X)$. So, the unary word $b^{|s|_b}$ is a common subword of X and we obtain $|u_X| \geq \left|b^{|s|_b}\right| = |s|_b \geq |s|/\sigma(X) = \mathrm{lcucs}(X)/\sigma(X)$, what we wanted.

• Existence of a polynomial approximation algorithm with bound $l/\log l$.

After the general method described in [13], the following algorithm gives in polynomial time an approximation with bound $l/\log l$ for LCS (resp. LCUS, resp. LCCS, resp. LCUCS).

1. Select a word $w \in X$ of minimal length l and factorize it under the form $w = y_1y_2\ldots y_p$ where
   - p is an integer smaller than or equal to $l/\log l$,
   - for any $i \in [1, p]$, y_i is a word of length $|y_i| = O(\log l)$.

2. For each $i \in [1, p]$, enumerate the subwords (resp. U-subwords, resp. C-subwords, resp. UC-subwords) of y_i, and memorize those that are solution of our problem.
3. Among the words selected at Step 2, return one of maximal length.

$\square$

Lemma 4. *Let w be a word that can be written as the concatenation of n strictly increasing words. Then, any* U-*subword (resp.* UC-*subword) of w that is strictly increasing and whose length is at least $n+1$ (resp. $n+2$), is a subword (resp.* C-*subword) of w.*

Proof. We divide our proof in two parts depending on the type of subword.

• Let s be a strictly increasing U-subword of w such that $|s| \geq n+1$. We show that s is a subword of w.

By hypothesis, either s is a subword of w or $\tilde{s}$ is subword of w. So, it suffices to demonstrate that the latter eventuality is forbidden. By contradiction, assume that $\tilde{s}$ is a subword of w. Then, one can write $\tilde{s}$ as the concatenation of n strictly increasing words. As $\tilde{s}$ is strictly decreasing, each of these n words has length of at most 1. It follows that the length of $\tilde{s}$ is at most n, which contradicts the hypothesis $|s| \geq n+1$.

• Let t be a strictly increasing UC-subword of w such that $|t| \geq n+2$ and let us prove that t is a C-subword of w.

By hypothesis, t is a U-subword of a some conjugate w' of w. Any conjugate of w may be written as the concatenation of (at most) $n+1$ strictly increasing words. So, the previous point applies with $n+1$ instead of n and w' instead of w. It yields that t is a subword of w' and thus, a C-subword of w. $\square$

Let S be a finite subset of $\mathbb{N}$. We consider implicitly that any integer is a symbol, and S as an alphabet. We denote by $\bar{S}$ the unique word over S that is strictly increasing and has length $\#S$.

It is shown in [5] that LCS_D is W[2]-hard for parameter k. In the following theorem, we state a somehow weaker result, LCS_D is W[1]-hard w.r.t. k, but exhibit a parameter preserving PTAS-reduction that is simpler than the one in [5] and can be adapted to LCUS_D.

Theorem 3. *Let $b : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to [1, \infty[$.*

- *The problems LCS_D and LCUS_D are* W[1]*-hard for parameter k.*
- *If either* LCS *or* LCUS *admit a polynomial approximation algorithm with bound $b\left(\min_{x \in X} |x|, \sigma(X), \#X\right)$ then* 2-MISH *admits a polynomial approximation algorithm with bound $b(|G|, |G|, |G|^2)$.*

Proof. First, we describe how we transform a graph G into an instance of LCS (resp. LCUS). Let G be a graph over $[1, |G|]$. To any edge $e \in \mathrm{E}(G)$, we associate a sequence denoted $x_{G,e}$ and defined by

$$x_{G,e} := \left(\overline{[1, |G|] \setminus e}\right) \bar{e}[2]\, \bar{e}[1] \left(\overline{[1, |G|] \setminus e}\right) .$$

In $x_{G,e}$, the prefix and suffix $\overline{[1, |G|] \setminus e}$ is the ordered set of vertices except the ones linked by e. In between, the latter are written in reverse alphabet order ($\bar{e}[2]\bar{e}[1]$). We can now define the instance X_G of LCS or (resp. LCUS).

$$X_G := \{x_{G,e}\}_{e \in \mathrm{E}(G)} \cup \left\{ \left(\overline{[1, |G|]}\right)^p \right\}_{p \in \left[1, |G|^2 - \#\mathrm{E}(G)\right]} .$$

In addition to the $x_{G,e}$'s ($e \in \mathrm{E}(G)$), X_G contains several powers of the string $\overline{[1, |G|]}$, which represents the whole ordered set of vertices of G. Actually, only $\overline{[1, |G|]}$ to the power one is useful to force the increasingness of common subwords of X_G, while the other powers allow us to set the cardinal of X_G to what we need.

Our proof relies on the four following properties of X_G.

P1. Given a graph G on $[1, |G|]$, the language X_G can clearly be computed in polynomial time.
P2. The shortest word of X_G is $\overline{[1, |G|]}$ and thus, $\min_{x \in X_G} |x| = \sigma(X_G) = |G|$. Moreover $\#X_G = |G|^2$.
P3. For any independent set I of G, $\bar{I}$ is a common subword of X_G of length $\#I$.
P4. Let s be a common U-subword s of X_G of length at least 5. The set of letters occurring in s is an independent set of G of cardinality $|s|$.

Let us prove Properties P3 and P4, starting with Property P3. Let I be an independent set of G. Since I is a subset of $\mathrm{V}(G) = [1, |G|]$, $\bar{I}$ is a subword of each $\overline{[1, |G|]}^p$ ($p \in \mathbb{N} \setminus \{0\}$). Let $e \in \mathrm{E}(G)$. Since I is an independent set, $\bar{e}[1]$ and $\bar{e}[2]$ cannot be both in I. If none occurs in I then $\bar{I}$ is a subword of $\overline{[1, |G|] \setminus e}$, and of $x_{G,e}$. Assume $\bar{e}[1]$ belongs to I and say it occurs in $\bar{I}$ at position i. Then, the prefix of length $i-1$ and the suffix of length $\left|\bar{I}\right| - i$ of $\bar{I}$ are both subwords of $\overline{[1, |G|]} \setminus e$. As $\bar{e}[1]$ can be picked up in $(\bar{e}[2]\bar{e}[1])$, we obtain that $\bar{I}$ is a subword of $x_{G,e}$. The case where $\bar{e}[2] \in I$ is symmetric. This completes the proof of Property P3.

We now demonstrate Property P4. Let s be a common U-subword of X_G of length at least 5. As either s or $\tilde{s}$ is a subword of $\overline{[1, |G|]}$, s is either strictly increasing or strictly decreasing. Since the same letters occur in s and $\tilde{s}$, we can, if necessary change s in $\tilde{s}$ to assume that s is strictly increasing. We know that s is a five letter U-subword of $x_{G,e}$. Note that $x_{G,e}$ can be written as the concatenation of four strictly increasing words: $\overline{[1, |G|] \setminus e}$, $\bar{e}[2]$, $\bar{e}[1]$ and finally $\overline{[1, |G|] \setminus e}$. Since, $5 = 4+1$, we obtain by Lemma 4 that s is a subword of $x_{G,e}$. If in $x_{G,e}$, it picks $\bar{e}[1]$ it cannot include $\bar{e}[2]$, and conversely. So two vertices linked by an edge cannot both occur together in s. This shows that s corresponds to an independent set of G. This concludes the proof of Property P4.

With these properties in hand, we now show the validity of our reduction, as well as the hardness and approximability results.
• The mapping that transforms any graph G over $[1, |G|]$ and any integer $k \in \mathbb{N}$, into an instance of LCS_D (resp. LCUS_D):

$$(G, k) \longmapsto \begin{cases} (X_G, k) & \text{si } k \geq 5 \\ \left(\left\{1^{\#J_G^5}\right\}, k\right) & \text{si } k < 5 \end{cases}$$

– is a valid reduction of 2-MISH$_D$ to LCS_D (resp. LCUS_D) by Properties P3 and P4,
– is computable polynomial time by Property P1 and Lemma 3,
– and preserves parameter k.

It allows us to establish the W[1]-hardness for parameter k of LCS_D (resp. LCUS_D).
• Assume it exists a polynomial approximation algorithm with bound $b\left(\min_{x \in X} |x|, \sigma(X), \#X\right)$ for LCS (resp. LCUS). Let us exhibit a polynomial approximation algorithm with bound $b\left(|G|, |G|, |G|^2\right)$ for 2-MISH.

Let G be the input graph. The algorithm proceeds as follows:

1. Compute language X_G,
2. Compute a common subword (resp. U-subword) s of X_G of length at least $\mathrm{lcs}(X_G)/b\left(\min_{x \in X_G} |x|, \sigma(X_G), \#X_G\right)$ (resp. $\mathrm{lcus}(X_G)/b\left(\min_{x \in X_G} |x|, \sigma(X_G), \#X_G\right)$),
3. If the length of s is at least 5, then return the set I of letters occurring in s,

4. Otherwise return J^5_G.

With a similar argument to the one developed in the proof of Theorem 2, one obtains the claimed bound. □

Theorem 4. *Let* $b : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to [1, \infty[$.

- *The problems* LCCS_D *and* LCUCS_D *are* W[1]*-hard for parameter* k.
- *If either* LCCS *or* LCUCS *admit a polynomial approximation algorithm with bound* $b\left(\min_{x \in X} |x|, \sigma(X), \#X\right)$ *then* 3-MISH *admits a polynomial approximation algorithm with bound* $b(|H|, |H|, |H|^3)$.

Proof. The proof is only slightly different from the one of Theorem 3. Given a 3-uniform hypergraph H on $[1, |H|]$, we set

$$x_{H,h} := \left(\overline{[1, |H|] \setminus e}\right)^2 \bar{h}[3] \left(\overline{[1, |H|] \setminus e}\right)^2 \bar{h}[2] \left(\overline{[1, |H|] \setminus e}\right)^2 \bar{h}[1]$$

for every hyperedge $h \in \mathrm{E}(H)$ and

$$X_H := \{x_{H,h}\}_{h \in \mathrm{E}(H)} \cup \left\{ \left(\overline{[1, |H|]}\right)^p \right\}_{p \in \left[1, |H|^3 - \#\mathrm{E}(H)\right]} .$$

As for Theorem 3, our proof relies on four properties of X_H.

P1. Given a 3-uniform hypergraph H on $[1, |H|]$, the language X_H can clearly be computed in polynomial time.
P2. $\min_{x \in X_H} |x| = \sigma(X_H) = |H|$ and $\#X_H = |H|^3$.
P3. For any independent set I of H, $\bar{I}$ is a common subword of X_H of length $\#I$.
P4. Let s be a common UC-subword of X_H of length at least 11. Then, the set of letters occurring in s is an independent set of H of cardinality $|s|$.

The remaining of the proof goes along the same lines as the one from Theorem 3 and are left to the reader. □

Theorem 3 (resp. Theorem 4) and results mentioned in Section 2.3 guarantee that for any constant $\delta \in [0, 1[$, one may not find a polynomial approximation algorithm for LCS or LCUS (resp. for LCCS or LCUCS) with bound $\max\{\min_{x \in X} |x|^\delta, \sigma(X)^\delta, (\#X)^{\delta/2}\}$ (resp. $\max\{\min_{x \in X} |x|^\delta, \sigma(X)^\delta, (\#X)^{\delta/3}\}$).

5 Conclusion

Our investigation provides the first hardness and approximability results concerning LCS for cyclic and unoriented strings. Similar results are also achieved for MISH in r-uniform hypergraphs. Nevertheless, some questions remain open.

• It is shown in [5] that LCS_D is W[1]-complete for parameter $(\#X, k)$, but the complexity of LCUS, LCCS and LCUCS for this parameter is unknown.

• The existence of a real valued constant δ, $0 \leq \delta < 1$, such that one problem among LCS, LCUS, LCCS or LCUCS admits a polynomial approximation algorithm with bound $(\#X)^\delta$ is open.

• For any $\sigma \in \mathbb{N}$, we demonstrated (Proposition 1) that problems LCS, LCUS, LCCS and LCUCS restricted to instances for which $\sigma(X) \leq \sigma$ admit a polynomial approximation algorithm with bound σ, but the existence of a PTAS require further studies.

• The problem 2-MISH gave rise to numerous publications, some of which are referenced in here, but fewer works concern MISH and r-MISH with $r \in \mathbb{N} \setminus \{0, 1, 2\}$. Specifically, the existence of a $t \in \mathbb{N} \setminus \{0\}$ such that 3-MISH_D parameterized by k is W[t]-complete is open, as is the existence of a polynomial approximation algorithm for 3-MISH with bound $o(|G| / \log |G|)$.

References

1. M. D. Adams, J. M. Kelley, J. D. Gocayne, M. Dubnick, M. H. Polymeropoulos, H. Xiao, C. R. Merril, A. Wu, B. Olde, R. F. Moreno, et al. Complementary DNA sequencing: expressed sequence tags and human genome project. *Science*, 252(5013):1651–1656, June 1991.
2. B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson. *Molecular Biology of the Cell*. Garland, New York, 1983.
3. A. Apostolico and Z. Galil, editors. *Pattern Matching Algorithms*. Oxford University Press, 1997.
4. 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 (CABIOS)*, 11(1):49–57, 1995.
5. 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(1–2):31–54, 1995.
6. R. Boppana and M. M. Halldòrsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32(2):180–196, 1992.
7. H. Bunke and U. Buehler. Applications of approximate string matching to 2D shape recognition. *Pattern Recognition*, 26(12):1797–1812, december 1993.
8. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, second edition, 2001.
9. M. Crochemore, G. M. Landau, and M. Ziv-Ukelson. A sub-quadratic sequence alignment algorithm for unrestricted cost matrices. In *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics*, pages 679–688, New York, 2002. ACM Press.
10. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
11. L. Engebretsen and J. Holmerin. Towards optimal lower bounds for clique and chromatic number. *Theoretical Computer Science*, 299(1–3):537–584, 2003.
12. Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Computer Science and Computational Biology. Cambridge University Press, 1997.
13. M. M. Halldòrsson. Approximations of weighted independent set and hereditary subset problems. *Journal of Graph Algorithms and Applications*, 4(1), 2000.
14. J. Hàstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.
15. T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal on Computing*, 24(5):1122–1139, 1995.
16. M. Krivelevich and B. Sudakov. Approximate coloring of uniform hypergraphs. *Journal of Algorithms*, 49(1):2–12, 2003.
17. M. Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, 35(2):73–78, 1990.
18. D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the Association for Computing Machinery*, 25(2):322–336, 1978.
19. W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.
20. K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003.
21. D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison*. CSLI Publications, second edition, 1999.
22. J. P. Schmidt. All highest scoring paths in weighted grid graphs and its application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27(4):972–992, 1998.