Longest Common Subsequence on Weighted Sequences

Evangelos Kipouridis
Basic Algorithms Research Copenhagen (BARC), University of Copenhagen, Denmark
kipouridis@di.ku.dk

Kostas Tsichlas
Computer Engineering and Informatics Department, University of Patras, Greece
ktsichlas@ceid.upatras.gr

Abstract

We consider the general problem of the Longest Common Subsequence (LCS) on weighted sequences. Weighted sequences are an extension of classical strings, where in each position every letter of the alphabet may occur with some probability. Previous results presented a $PTAS$ and noticed that no $FPTAS$ is possible unless $P = NP$. In this paper we essentially close the gap between upper and lower bounds by improving both. First of all, we provide an $EPTAS$ for bounded alphabets (which is the most natural case), and prove that there does not exist any $EPTAS$ for unbounded alphabets unless $FPT = W[1]$. Furthermore, under the Exponential Time Hypothesis, we provide a lower bound which shows that no significantly better $PTAS$ can exist for unbounded alphabets. As a side note, we prove that it is sufficient to work with only one threshold in the general variant of the problem.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → W hierarchy; Theory of computation → Problems, reductions and completeness

Keywords and phrases WLCS, LCS, weighted sequences, approximation algorithms, lower bound

Digital Object Identifier 10.4230/LIPIcs.CPM.2020.19

Related Version Full version available at arXiv: https://arxiv.org/abs/1901.04068.

Funding *Evangelos Kipouridis*: Thorup's Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation, and European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 801199.

Acknowledgements We would like to thank the anonymous reviewers for their careful reading of our paper and their many insightful comments and suggestions.

1 Introduction

1.1 General concepts

We consider the problem of determining the LCS (Longest Common Subsequence) on weighted sequences. Weighted sequences, also known as p-weighted sequences or Position Weighted Matrices (PWM) [3, 36] are probabilistic sequences which extend the notion of strings, in the sense that in each position there is some probability for each letter of an alphabet Σ to occur there.

Weighted sequences were introduced as a tool for motif discovery and local alignment and are extensively used in molecular biology [23]. They have been studied both in the context of short sequences (binding sites, sequences resulting from multiple alignment, etc.) and on large sequences, such as complete chromosome sequences that have been obtained

31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020).
Editors: Inge Li Gørtz and Oren Weimann; Article No. 19; pp. 19:1–19:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

using a whole-genome shotgun strategy [32, 37]. Weighted sequences are able to keep all the information produced by such strategies, while classical strings impose restrictions that oversimplify the original data.

Basic concepts concerning the combinatorics of weighted sequences (like pattern matching, repeats discovery and cover computation) were studied using weighted suffix trees [26], Crochemore's partitioning [9, 11, 18], the Karp-Miller-Rabin algorithm [18], and other approaches [43, 30]. Other interesting results include approximate and gapped pattern matching [6, 41, 34], online pattern matching [16], weighted indexing [2, 10], swapped matching [40], the all-covers and all-seeds problem [39, 42], extracting motifs [28], and the weighted shortest common supersequence problem [4, 17]. There are also some more practical results on mapping short weighted sequences to a reference genome [7] (also studied in the parallel setting [27]), as well as on the reporting version of the problem which we also consider in this paper [11].

The Longest Common Subsequence (*LCS*) problem is a well-known measure of similarity between two strings. Given two strings, the output should be the length of the longest subsequence common to both strings. Dynamic programming solutions [25, 38] for this problem are classical textbook algorithms in Computer Science. *LCS* has been applied in computational biology for measuring the commonality of DNA molecules or proteins which may yield similar functionality. A very interesting survey on algorithms for the *LCS* can be found in [13]. The current *LCS* algorithms are considered optimal, since matching lower bounds (under the Strong Exponential Time Hypothesis) were proven [1, 14].

Extensions of this problem on more general structures have also been investigated (trees and matrices [5], run-length encoded strings [8], and more). One interesting variant of the *LCS* is the Heaviest Common Subsequence (*HCS*) where the matching between different letters is assigned a different weight, and the goal is to maximize the weight of the common subsequence, rather than its length.

1.2 Weighted LCS

The problem studied in this paper is the weighted *LCS* (WLCS) problem. It was introduced by Amir et al. [3] as an extension of the classical *LCS* problem on weighted sequences. Given two weighted sequences, the goal is to find a longest string which has a high probability of appearing in both sequences. Amir et al. initially solved an easier version of this problem in polynomial time, but unfortunately its applications are limited. As far as the general problem is concerned, they hinted its NP-Hardness by giving an NP-Hardness result on a closely related problem, which they call the log-probability version of WLCS. In short, the problem is the same, but all products in its definition are replaced with sums. Their proof is based on a Turing reduction and only works for unbounded alphabets. Finally, Amir et al. provide an $\frac{1}{|\Sigma|}$-approximation algorithm for the WLCS problem.

Cygan et al. [19] strengthened the evidence that WLCS is NP-Hard by providing an NP-Completeness result on the decision log-probability version of WLCS (informally introduced in the previous paragraph), already for alphabets of size 2, using a Karp reduction; for alphabets of size 1 the solution is trivial since there is no uncertainty. They also gave an $\frac{1}{2}$-approximation algorithm and a *PTAS*, while also noticing that an *FPTAS* cannot exist, assuming WLCS is indeed NP-Hard, as hinted by their evidence, and that P $\neq$ NP. Finally, they proved that every instance of the problem can be reduced to a more restricted class of instances. However, for this to be achieved their algorithm needs to perform exact computations of roots and logarithms that may make the algorithm to err.

Finally, it is worth noting that Charalampopoulos et al. [17], proved that unless P=NP, WLCS cannot be solved in $\mathcal{O}(n^{f(a)})$ time, for any function $f(a)$, where a is the cut-off probability. We note that this result concerns exact computations rather than approximations.

1.3 Our results

In this paper we essentially close the gap between upper and lower bounds for WLCS by improving both; we prove that the problem is indeed NP-Hard even for alphabets of size 2. Furthermore, we provide an $EPTAS$ for bounded alphabets. These two results, along with the $FPTAS$ observation by Cygan et al. completely characterize the complexity of WLCS for bounded alphabets. For unbounded alphabets, a $PTAS$ was already known by Cygan et al. [19]. We show matching lower bounds, both by ruling out the possibility of an $EPTAS$, and by showing that, under the Exponential Time Hypothesis, no significantly better $PTAS$ can exist. We also prove that every instance of WLCS can be reduced to a restricted class of instances without using roots and logarithms, thus being able to actually achieve exact computations without rounding errors that can make the algorithm err.

As noted in the previous paragraph, apart from essentially closing the gap between hardness results and faster algorithms we also circumvent the need to work with roots and logarithms as the previous results did. In short, by taking advantage of the property that $(ab)^c = a^c b^c$ and setting c to be an appropriate logarithm, previous results transformed any instance to a more manageable form. However, this transformation introduces an error that may make the algorithm err as noted in the full version of the paper [29]. Table 1 summarizes the above discussion. Table 2 summarizes our results depending on the alphabet-size.

A short discussion is in order with respect to what new insights on weighted LCS enabled us to achieve progress. Our most crucial observation is the fact that the problem behaves differently in the natural case of a bounded alphabet, and in the case of an unbounded alphabet. Without this distinction, closing the gap between upper and lower bounds was unlikely. That's because, on the one hand, no $EPTAS$ for the general case could be found, as none existed. On the other hand, proving that no $EPTAS$ exists requires reductions that work only on unbounded alphabets. The aforementioned distinction is what enabled us to understand that modifying the existing reductions, which work for alphabets of size 2, would be futile in proving $W[1]$-Hardness.

Furthermore, it was crucial to identify that working with products is the core difficulty in proving NP-Hardness of weighted LCS. The introduction of the log-probability version of the weighted LCS reflects the assumption that the difference between working with sums and working with products is just a technicality. After [3] and [19] successfully proved NP-Hardness for the log-probability version, it was natural to attempt reducing from it for proving NP-Hardness of the weighted LCS problem. Despite the apparent similarities between the two problems, their difference did not allow us to craft such a reduction. For the same reason, Cygan et al. used a model that assumed infinite precision computations with reals, while we are able to avoid such a strong assumption.

1.4 Organization of the paper

The rest of the paper is organized as follows. In Section 2, we provide necessary definitions and discuss the model of computation. In Section 3, we show that WLCS is NP-Complete while in Section 4, we provide the $EPTAS$ algorithm for bounded alphabets, which is also an improved $PTAS$ for unbounded alphabets. In Section 5, we show that there can be no $EPTAS$ for unbounded alphabets by showing that this problem is $W[1]$-hard and in Section 6, we describe the matching conditional lower bound. We conclude in Section 7.

Due to space constraints, some proofs and technical discussions are only available on the full version of the paper [29]. More specifically, in the full version we present an algorithm that transforms any instance of our problem to an equivalent, but easier to handle, instance. We also show that the rounding errors introduced by working with reals (logarithms and roots) may cause a similar algorithm by Cygan et al. [19] to err if standard rounding is used.

Table 1 Results on WLCS.

	Amir et al.	Cygan et al.	Our results
NP-Hardness of WLCS	Hinted, by NP-Hardness of Log-probability version (Turing reduction - only for unbounded alphabets)	Hinted, by NP-Hardness of Log-probability version (Karp reduction - already from alphabets size 2)	Proved (Karp Reduction - already from alphabets of size 2)
Approximation Algorithms	$\frac{1}{\Sigma}$-Approximation	$PTAS$	$EPTAS$ for bounded alphabets, Improved $PTAS$ for unbounded
Proof that no $EPTAS$ exists for unbounded alphabets	No	No	Yes
Lower bound on any $PTAS$	No	No	Matching the upper bound, under ETH
Reduction to a restricted class of instances	No	Yes, by assuming exact computations of logarithms	Yes, without any extra assumptions

Table 2 Results depending on the Alphabet Size.

Alphabet Size	Previous Results	Our results
1	Trivial	Trivial
Constant Size	No $FPTAS$ possible	Achieved $EPTAS$
Depending on the input	Achieved $PTAS$	No $EPTAS$ possible, Improved $PTAS$, Matching Lower Bound

2 Preliminaries

2.1 Basic Definitions

Let $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_K\}$ be a finite alphabet. We deal both with bounded ($K = O(1)$) and unbounded alphabets. Σ^d denotes the set of all words of length d over Σ. Σ^* denotes the set of all words over Σ.

▶ **Definition 1** (Weighted Sequence). *A weighted sequence X is a sequence of functions $p_1^{(X)}, \dots, p_{|X|}^{(X)}$, where each function assigns a probability to each letter from Σ. We thus have $\sum_{j=1}^{K} p_i^{(X)}(\sigma_j) = 1$ for all i, and $p_i^{(X)}(\sigma_j) \geq 0$ for all i, j.*

By $WS(\Sigma)$ we denote the set of all weighted sequences over Σ. Let $X \in WS(\Sigma)$. Let $Seq_d^{|X|}$ be the set of all increasing sequences of d positions in X. For a string $s \in \Sigma^d$ and $\pi \in Seq_d^{|X|}$, define $P_X(\pi, s)$ as the probability that the subsequence on positions corresponding to π in X equals s. More formally, if $\pi = (i_1, i_2, \dots, i_d)$ and s_k denotes the k-th letter of s, then

$$P_X(\pi, s) = \prod_{k=1}^{d} p_{i_k}^{(X)}(s_k)$$

Denote

$$SUBS(X,a) = \{s \in \Sigma^* | \exists \pi \in Seq_{|s|}^{|X|} \textit{ such that } P_X(\pi, s) \geq a\}$$

That is, $SUBS(X,a)$ is the set of deterministic strings which match a subsequence of X with probability at least a. Every $s \in SUBS(X,a)$ is called an a-subsequence of X.

Let us give a clarifying example. If $\Sigma = \{\sigma_1, \sigma_2\}$ and X is a long weighted sequence, where in each position the probability for each letter to appear is 0.5, then $SUBS(X, 0.3)$ does not contain $s = \sigma_1\sigma_1$, as, for any increasing subsequence of 2 positions, the probability of s appearing is $0.25 < 0.3$.

The decision problem we consider is the following:

▶ **Definition 2** ((a_1, a_2)-WLCS decision problem). *Given two weighted sequences X, Y, two cut-off probabilities a_1, a_2 and a number k, find if the longest string s contained in $SUBS(X, a_1) \cap SUBS(Y, a_2)$ has length at least k.*

Naturally, the respective optimization problem is the following:

▶ **Definition 3** ((a_1, a_2)-WLCS optimization problem). *Given two weighted sequences X, Y, and two cut-off probabilities a_1, a_2, find the length of the longest string contained in $SUBS(X, a_1) \cap SUBS(Y, a_2)$.*

Both in the decision and the optimization version, the WLCS problem is the (a_1, a_2)-WLCS problem, where $a_1 = a_2$. We denote these (equal) probabilities by a ($a = a_1 = a_2$) for concreteness.

Let us note that the problem is only interesting if $|\Sigma| \geq 2$. For $|\Sigma| = 1$ the problem is trivial since there is no uncertainty at all. The same letter appears in every position in both strings with probability 1, and thus the answer is simply the length of the shorter weighted sequence.

Finally, let us also state that the Log-Probability version of the WLCS, studied in previous papers, is the same as the original WLCS if we replace $P_X(\pi, s) = \prod_{k=1}^{d} p_{i_k}^{(X)}(s_k)$ by $P_X(\pi, s) = \sum_{k=1}^{d} p_{i_k}^{(X)}(s_k)$.

2.2 Model of Computation

Our model of computation is the standard word RAM, introduced by Fredman and Willard [20] to simulate programming languages like C. The word size is $w = \Omega(\log I)$, where I is the input size in bits, so as to allow random access indexing of the whole input. Thus, arithmetic operations between words take constant time. However, due to the nature of our problem, it is necessary to compute products of many numbers. This can produce numbers that are much larger than the word size. We even allow numbers in the input to be larger than 2^w (these numbers just need to use more than one word to be represented). We generally assume that each number in the input is represented by at most B bits, but we do not pose any constraint on B other than the trivial one that $B < I$. Of course, in cases where we deal with numbers that occupy many words, we no longer have unit-cost arithmetic operations; we guarantee, however, that our results only use linear or near-linear time operations (like comparisons and multiplications) on numbers polynomial in the input size. Thus, although we do not enjoy the unit-cost assumption for arbitrary numbers, we always stay within the polynomial-time regime.

2.3 Basic Operations

In this subsection we discuss the multiplication of two B-bit input numbers in (polynomial) $Mul_w(B)$ time, where w is the word-size. For example, for integers there exists a multiplication algorithm by Harvey and van der Hoeven [24] with time complexity $Mul_w(B) = \mathcal{O}(B \log B)$ (generally the running time can also depend on w, although in this case it does not). Let us notice that although the result is unpublished yet, we use it due to its easy to read time complexity; it is trivial to use other algorithms instead, like the one from Fürer [21], or the more practical one by Schönhage and Strassen [35]. We establish the complexity of multiplying x B-bit numbers. Our divide and conquer algorithm splits the numbers into two (equal sized) groups, recursively multiplies each, and multiplies the results in $Mul_w\left(\frac{xB}{2}\right)$ time. By a direct application of the Master Theorem by Bentley et al. [12] we prove the following lemma.

▶ **Lemma 4.** *Multiplying x B-bit numbers costs*
- $\mathcal{O}(Mul_w(xB)\log(xB))$ *time if* $Mul_w(xB) = \Theta(xB\log^k(xB))$ *for some constant* k,
- $\mathcal{O}((xB)^c)$ *else if* $Mul_w(xB) = \mathcal{O}((xB)^c)$ *for some constant* $c \geq 1$,

assuming that $Mul_w(N)$ is a polynomial time algorithm that multiplies two N-bit numbers.

Proof. The algorithm simply splits the numbers in two equal-sized groups, recursively multiplies each, and then multiplies the results. Let $N = xB$. We have that the running time for multiplying x B-bit numbers is $T(N) = 2T(\frac{N}{2}) + Mul_w(N)$. Since $c_{crit} = \log_2 2 = 1$, and $Mul_w(N) = \Omega(N)$, the Master Theorem [12] gives two cases. Either $Mul_w(N) = \Theta(N\log^k(N))$ for some constant k, in which case $T(N) = \mathcal{O}(Mul_w(N)\log N)$, or else $Mul_w(N) = \mathcal{O}(N^c)$ for some constant $c \geq 1$ (such a constant exists since we assume polynomial time multiplications). In this case, since it holds that $2Mul_w(\frac{N}{2}) \leq 2Mul_w(N)$, we get that $T(N) = Mul_w(N)$ if $c > c_{crit} = 1$. Notice that we handled all cases, since $Mul_w(N) = N$ is handled by the first case with $k = 0$, and whatever does not fit in the first case, definitely fits in the second, since we assumed that $Mul_w(N)$ is polynomial in N. ◀

▶ **Corollary 5.** *Multiplying x B-bit numbers costs polynomial time by using any polynomial time algorithm for multiplying two B-bit numbers as a black box. Especially if we use Harvey and Van Der Hoeven's algorithm, the time cost is $\mathcal{O}\left(xB\log^2(xB)\right)$.*

Let us also notice that the way to divide two B-bit numbers is simply storing both the numerator and the denominator. Comparing two numbers $x_1 = \frac{num_1}{den_1}$ and $x_2 = \frac{num_2}{den_2}$ can be done by comparing $num_1 \times den_2$ and $num_2 \times den_1$. The only other operation we need when working with such fractions is subtracting a B-bit number $x = \frac{num}{den}$ from 1. This is simply $\frac{den-num}{den}$.

3 NP-Completeness

An NP-Completeness proof for the integer log-probability version of the WLCS problem has been given in [19]. This is a closely related problem, with the main difference being that products are replaced with sums. We do not know of any way to reduce from this log-probability version to WLCS other than exponentiating. As stated in the explanation of our model of computation in Section 2, there is no limit on the number of bits needed to represent a single number (it just occupies a lot of words). This means that, if the input consisted of I bits, and there was a number (probability) represented with $\frac{I}{100}$ bits, exponentiating would result in a number with $2^{\frac{I}{100}}$ bits, meaning the reduction would not be a polynomial-time one. For this reason, we believe that although it is easier to prove NP-Completeness for the integer log-probability version of the problem, there is no easy way to use it for proving NP-Completeness for the general version. We, thus, give a reduction from the NP-Complete problem Subset Product [22] which proves NP-Completeness directly for the general problem.

Notice that for alphabets consisting of one letter, the problem is trivial since there is no uncertainty at all. In the following, we prove that even for alphabets consisting of two letters, the problem is NP-Complete.

▶ **Definition 6** (Subset Product). *Given a set L of n integers and an integer P, find if there exists a subset of the numbers in L with product P.*

▶ **Lemma 7.** *WLCS is NP-Complete, even for alphabets of size 2.*

Proof. Obviously $WLCS \in NP$ since the increasing subsequences π_1, π_2 and the string s for which $P_X(\pi_1, s) \geq a, P_Y(\pi_2, s) \geq a$ are a certificate which, along with the input, can be used to verify in polynomial time that the problem has a solution.

Let (L, P) be an instance of Subset Product and let $n = |L|$. By L_i we denote the i-th number of the set L, assuming any fixed ordering of the n numbers of L. We give a polynomial-time reduction to a (X, Y, a, k) instance of WLCS, with alphabet size 2 (we call the letters $'A'$ and $'B'$).

The core idea is the following: The weighted sequences have n positions (plus 2 more for technical reasons related to the threshold a). The number k is equal to the length of the sequences, meaning that we pick every position, and the only question is whether we picked letter $'A'$ or letter $'B'$. Letter $'A'$ in position i corresponds to picking the i-th number in the original Subset Product, while letter $'B'$ corresponds to not picking it. Finally, the letters $'A'$ picked in X form an inequality of the form: "some product is $\geq P$", while the same letters in Y form the inequality: "the same product is $\leq P$". For these two to hold simultaneously, it must be the case that we found some product equal to P, which is the goal of the original Subset Product.

More formally, the weighted sequences have size $n + 2$. Let $c_i = \frac{1}{1+L_i}$ and $d_i = \frac{1}{1+\frac{1}{L_i}}$.

$$p_i^{(X)}('A') = c_i L_i, 1 \leq i \leq n \qquad p_i^{(Y)}('A') = \frac{d_i}{L_i}, 1 \leq i \leq n$$

$$p_{n+1}^{(X)}('A') = 1 \qquad p_{n+1}^{(Y)}('A') = \prod_{j=1}^{n} \frac{1}{L_i} = \frac{\prod_{j=1}^{n} c_i}{\prod_{j=1}^{n} d_i}$$

$$p_{n+2}^{(X)}('A') = \frac{1}{P^2} \qquad p_{n+2}^{(Y)}('A') = 1$$

where $p_i^{(X)}('B') = 1 - p_i^{(X)}('A')$ for all i, and similarly for Y. Notice that, in particular, $p_i^{(X)}('B') = c_i, 1 \leq i \leq n$ and $p_i^{(Y)}('B') = d_i, 1 \leq i \leq n$. Finally, we set $k = n + 2$ and $a = \frac{\prod_{j=1}^{n} c_i}{P}$.

First of all, notice that since we must find a string of length $n + 2$, we must choose a letter from every position. Thus, a choice of letter at some position on X corresponds to the same choice of letter at that position on Y. The choice of letter on positions $n + 1$ and $n + 2$ is $'A'$ in both cases since

$$p_{n+1}^{(X)}('B') = p_{n+2}^{(Y)}('B') = 0$$

Suppose that the numbers at positions $\{i_1, \ldots, i_\ell\}$ give product P:

$$\prod_{j=1}^{\ell} L_{i_j} = P$$

Then, we form the string s by picking $'A'$ at positions $\{i_1, \ldots, i_\ell, n+1, n+2\}$ and $'B'$ at all other positions. Thus

$$P_X(\{1,2,\ldots,n+2\}, s) = \frac{\prod_{j=1}^{\ell} L_{i_j} \prod_{j=1}^{n} c_i}{P^2} = \frac{\prod_{j=1}^{n} c_i}{P} = a$$

$$P_Y(\{1,2,\ldots,n+2\}, s) = \frac{\prod_{j=1}^{n} d_i \prod_{j=1}^{n} c_i}{\prod_{j=1}^{\ell} L_{i_j} \prod_{j=1}^{n} d_i} = \frac{\prod_{j=1}^{n} c_i}{P} = a$$

Conversely, suppose a solution for the WLCS problem, where the string s is formed by picking $'A'$ at positions $\{i_1, \ldots, i_\ell, n+1, n+2\}$ and $'B'$ at all other positions. It holds that:

$$P_X(\{1,2,\ldots,n+2\}, s) = \frac{\prod_{j=1}^{\ell} L_{i_j} \prod_{j=1}^{n} c_i}{P^2} \geq a \implies \prod_{j=1}^{\ell} L_{i_j} \geq P$$

$$P_Y(\{1,2,\ldots,n+2\}, s) = \frac{\prod_{j=1}^{n} d_i \prod_{j=1}^{n} c_i}{\prod_{j=1}^{\ell} L_{i_j} \prod_{j=1}^{n} d_i} \geq a \implies \prod_{j=1}^{\ell} L_{i_j} \leq P$$

The above imply that $\prod_{j=1}^{\ell} L_{i_j} = P$. Finally, notice that all computations are done in polynomial time, due to Corollary 5. ◀

4 EPTAS for Bounded Alphabets, Improved PTAS for Unbounded Alphabets

We now give an Efficient Polynomial Time Approximation Scheme (*EPTAS*) for the case where our alphabet size is bounded ($|\Sigma| = O(1)$). Let us notice that this is the case when working with DNA sequences ($|\Sigma| = 4$), the most usual application of weighted sequences. The same algorithm is an improved (when compared to [19]) *PTAS* in the case of unbounded alphabets. This means that the WLCS problem is Fixed-Parameter Tractable for constant size alphabets and thus belongs to the corresponding complexity class *FPT* as shown in Corollary 11.

The authors in [19] first noted that there is no *FPTAS* unless $P = NP$, and so we can only hope for an *EPTAS*. Our result relies on their following result:

▶ **Lemma 8** (Lemma 4.6 of [19]). *It is possible to find, in polynomial time, a solution of size d to the WLCS optimization problem such that the optimal value OPT is guaranteed to be either d or $d+1$ (however we do not know which one holds).*

Their *PTAS* uses the above result and in case the approximation is guaranteed to be good enough ($d > (1-\epsilon)(d+1)$, which implies that $d > (1-\epsilon)OPT$), it stops. Else, it holds that $\frac{1}{\epsilon} \geq d+1 \geq OPT$, and the *PTAS* exhaustively searches all subsequences of X, all subsequences of Y, and all possible strings of length $d+1$, for a total complexity of

$$\mathcal{O}\left(Mul_w\left(\frac{B}{\epsilon}\right)\log\left(\frac{B}{\epsilon}\right)|\Sigma|^{\frac{1}{\epsilon}}\binom{n}{\frac{1}{\epsilon}}^2\right)$$

$Mul_w(\frac{B}{\epsilon})\log(\frac{B}{\epsilon})$ is the time needed to multiply $d+1$ numbers with at most B-bits each, by Lemma 4, and is insignificant compared to the other terms. Our *EPTAS* improves the exhaustive search part to

$$\mathcal{O}\left(Mul_w\left(\frac{B}{\epsilon}\right)\frac{n}{\epsilon}|\Sigma|^{\frac{1}{\epsilon}}\right)$$

which is polynomial in the input size, in case of bounded alphabets. The following lemma is needed.

▶ **Lemma 9.** *Given a weighted sequence X of length n, and a string s of length d, it is possible to find the maximum number a such that there exists an increasing subsequence π of length d for which $P_X(\pi, s) = a$. The running time of the algorithm is $O(Mul_w(dB)nd)$, where B is the maximum number of bits needed to represent each probability in X.*

Proof. We use dynamic programming. Let s_j be the string formed by the first j letters of s, c_j be the j-th letter of s and $opt_X(i, j)$ be the maximum number such that there exists an increasing subsequence π' of length j whose last term π'_j is at most i and for which $P_X(\pi', s_j) = opt_X(i, j)$. Since we choose whether c_j is picked from the i-th position of X, it holds that:

$$opt_X(i, j) = \max\{opt_X(i-1, j), opt_X(i-1, j-1)p_i^{(X)}(c_j)\}$$

For the base cases, $opt_X(i, 0) = 1$ for all i (we can always form the empty string with certainty, by not picking anything), and $opt_X(0, j) = 0$ for $j > 0$ (not picking anything never gives us a non-empty string). We are interested in the value $opt_X(|X|, |s|)$. ◀

Now we are ready to give our *EPTAS*.

▶ **Theorem 10.** *For any value $\epsilon \in (0, 1]$ there exists an $(1-\epsilon)$-approximation algorithm for the WLCS problem which runs in $\mathcal{O}\left(poly(I) + \frac{n}{\epsilon}Mul_w\left(\frac{B}{\epsilon}\right)|\Sigma|^{\frac{1}{\epsilon}}\right)$ time and uses $\mathcal{O}(poly(I))$ space, where I is the input size, $n = |X| + |Y|$ and B is the maximum number of bits needed to represent a probability in X and Y. Consequently, the WLCS problem admits an EPTAS for bounded alphabets.*

Proof. We begin by using Lemma 8 to find an a-subsequence of length d, such that the optimal solution is at most $d+1$. If $d+1 \geq \frac{1}{\epsilon}$, we are done, since in that case we have a $\frac{d}{d+1} = 1 - \frac{1}{d+1} \geq (1-\epsilon)$ approximation. Otherwise, we try all possible strings $s \in |\Sigma|^{d+1}$, and use Lemma 9 to check if any one of them can appear in both weighted sequences with probability at least a. ◀

▶ **Corollary 11.** *$WLCS \in FPT$ for bounded alphabets, parameterized by the solution length.*

Proof. Follows directly from [31], Proposition 2. ◀

5 No EPTAS for Unbounded Alphabets

We have already seen that there is no *FPTAS* for WLCS, even for alphabets of size 2, unless $P = NP$. We have also shown an *EPTAS* for bounded alphabets and a *PTAS* for unbounded alphabets. The natural question that arises is: Is it possible to give an *EPTAS* for unbounded alphabets?

We answer this question negatively, by proving that WLCS is $W[1]$-hard, meaning that it does not admit an *EPTAS* (and is in fact not even in *FPT*) unless $FPT = W[1]$ ([31], Corollary 1). To show this, we give a 2-step *FPT*-reduction from Perfect Code, which was shown to be $W[1]$-Complete in [15], to k-sized Subset Product and then to WLCS. The k-sized Subset Product problem is the Subset Product problem with the additional constraint that the target subset must be of size k.

▶ **Definition 12** (Perfect Code). *A perfect code is a set of vertices $V' \subseteq V$ with the property that for each vertex $u \in V$ there is precisely one vertex in $N_G(u) \cap V'$, where $N_G(u)$ is the set of adjacent nodes of u in G.*

In the perfect code problem, we are given an undirected graph G and a positive integer k, and we need to decide whether G has a k-element perfect code. Notice that the definition of a perfect code implies that there is a perfect code iff there is a set $V' \subseteq V$ for which $\bigcup_{u \in V'} N_G(u) = V$ and $N_G(u) \cap N_G(v) = \emptyset$ for all $u, v \in V', u \neq v$. First we show that k-sized Subset Product is $W[1]$-hard.

▶ **Lemma 13.** *k-sized Subset Product is $W[1]$-hard.*

Proof. Let $(G = (V, E), k)$ be an instance of Perfect Code. Suppose that the vertices are $V = \{1, \ldots, n\}$. First of all, we compute the first n prime numbers using the Sieve of Eratosthenes. We denote the i-th prime number as p_i. The set of positive integers $L = \{L_1, L_2, \ldots, L_n\}$ as well as the positive integer P are defined as follows:

$$L_v = \prod_{u \in N_G(v)} p_u, \; P = \prod_{v=1}^{n} p_v$$

Notice that due to the unique prime factorization theorem, a subset of k numbers from the set L have product P iff G has a k-element Perfect Code.

The size of our primes is $O(n \log n)$ due to the prime number theorem. Thus, they require $O(\log n)$ bits to be represented. Each integer in L, as well as in P, is computed using Corollary 5 in $O(n \log^3 n)$ time, for an overall $O(n^2 \log^3 n)$ complexity for our reduction. Since the new parameter k is the same as the old one (no dependence on n), our reduction is in fact an FPT-reduction. ◀

Our result for this section is the following.

▶ **Theorem 14.** *WLCS, parameterized by the length of the solution, is $W[1]$-hard.*

Proof. To prove the theorem we create diagonal weighted sequences. That is, we require each letter to appear only in one position and vice-versa. In this way, the subsequences picked for X and Y are the same. The above rule is broken by the addition of two auxiliary letters that are there to make the probabilities add up to 1 in each position. This creates no problem because we make sure that these letters are never picked. Finally, we force the product to be equal to our target, by forcing it to be at most our target and at least our target at the same time.

More formally, let $(L = \{L_1, L_2, \ldots, L_n\}, k, P)$ be an instance of the k-sized Subset Product problem and let $M = m^{k+1}$, where m is the maximum number in set L. Notice that if $m^k \leq P$ then we only need to check the product of the highest k numbers of L, which means the problem is solvable in polynomial time. Thus we can assume that $M \geq m^k > P$. The alphabet of X, Y is $\Sigma = \{1, 2, \ldots, n, n+1, n+2, n+3\}$ and we set $a = \frac{1}{PM^k}$.

$$p_i^{(X)}(i) = \frac{L_i}{M}, \; 1 \leq i \leq n \qquad p_i^{(Y)}(i) = \frac{1}{ML_i}, \; 1 \leq i \leq n$$

$$p_{n+1}^{(X)}(n+1) = \frac{1}{P^2} \qquad p_{n+1}^{(Y)}(n+1) = 1$$

$$p_i^{(X)}(n+2) = 1 - p_i^{(X)}(i), \; 1 \leq i \leq n+1 \qquad p_i^{(Y)}(n+3) = 1 - p_i^{(Y)}(i), \; 1 \leq i \leq n+1$$

All non-specified probabilities are equal to 0. Notice that symbols $n+2$ and $n+3$ are used to guarantee that probabilities sum up to 1.

We show that the instance $(X, Y, a, k+1)$ has a solution iff (L, k, P) has a solution. Suppose there exists a solution to (L, k, P). Then, there exists an increasing subsequence $\pi = (i_1, \dots, i_k)$ such that $\prod_{j=1}^{k} L_{i_j} = P$. Let π' be π extended by the number $i_{k+1} = n+1$ and s be the string $i_1 i_2 \dots i_{k+1}$. It holds that $P_X(\pi', s) = P_Y(\pi', s) = a$.

Conversely, suppose there exists a solution to $(X, Y, a, k+1)$. Then there exist increasing subsequences $\pi = (i_1, \dots, i_{k+1}), \pi' = (j_1, \dots, j_{k+1})$ and a string s such that $P_X(\pi, s) \geq a, P_Y(\pi', s) \geq a$. First of all, notice that, due to $p_i^{(X)}(n+3) = p_i^{(Y)}(n+2) = 0$ for all i, s does not contain letters $n+2$ and $n+3$, which leaves only one choice for every position. Also each letter appears only once in each sequence, and in the same position. Thus, $\pi = \pi'$, and due to our construction the i-th letter of s is the i-th member of π. Finally, not picking position $n+1$ would result in $P_Y(\pi, s) < a$ due to the fact that $P < M$. Thus, the last letter of s is $n+1$. It holds that:

$$P_X(\{i_1, \dots, i_{k+1}\}, s) \geq a \implies \frac{\prod_{i=1}^{k} L_{\pi_i}}{P^2 M^k} \geq \frac{1}{PM^k} \implies \prod_{i=1}^{k} L_{\pi_i} \geq P$$

$$P_Y(\{i_1, \dots, i_{k+1}\}, s) \geq a \implies \frac{1}{M^k \prod_{i=1}^{k} L_{\pi_i}} \geq \frac{1}{PM^k} \implies \prod_{i=1}^{k} L_{\pi_i} \leq P$$

The above two inequalities imply a k-sized subset of L with product equal to P.

The reduction is a polynomial-time one, due to Corollary 5. More than that, it is an FPT-reduction since the new parameter k is equal to the old parameter incremented by one, and thus has no dependence on n. ◀

6 Matching Conditional Lower Bound on any PTAS

In the d-SUM problem, we are given N numbers and need to decide whether there exists a d-tuple that sums to zero. Patrascu and Williams [33] proved that any algorithm for solving the d-SUM problem requires $n^{\Omega(d)}$ time, unless the Exponential Time Hypothesis (ETH) fails. To show this, they first proved a hardness result for a variant of 3-SAT, the sparse 1-in-3 SAT.

▶ **Definition 15** (Sparse 1-in-3 SAT). *Given a boolean formula with n variables and $O(n)$ clauses in 3 CNF form, where each variable appears in a constant number of clauses, determine whether there exists an assignment of the variables such that each clause is satisfied by exactly one variable.*

They first prove the following hardness result under ETH.

▶ **Proposition 16.** *Under ETH, there is an (unknown) constant s_3 such that there exists no algorithm to solve sparse 1-in-3 SAT in $\mathcal{O}(2^{\delta n})$ time for $\delta < s_3$.*

By assuming an $n^{\mathcal{O}(d)}$ time algorithm for d-SUM they disproved the above fact, which cannot happen under ETH. We use the same technique for proving an $n^{\Omega(k)}$ lower bound for k-sized Subset Product.

▶ **Lemma 17.** *Assuming the ETH, the problem of k-sized Subset Product cannot be solved in $\mathcal{O}(n^{\frac{s_3 k}{101}})$ time on instances satisfying $k < n^{0.99}$ and each number in the input set L has $\mathcal{O}(\log n(\log k + \log\log n))$ bits, where n is the size of L, and P is the target which can be arbitrarily big.*

Proof. Let f be a sparse 1-in-3 SAT instance with N variables and $M = \mathcal{O}(N)$ clauses, and $k > \frac{1}{s_3}$. Conceptually, we split the variables of f into k blocks of equal size - apart from the last block that may have smaller size. Each block contains at most $\lceil \frac{N}{k} \rceil$ variables, and thus there are at most $2^{\lceil \frac{N}{k} \rceil}$ different assignments of values to the group-of-variables within a block. For each block and for each one of these assignments we generate a number which serves as an identifier of the corresponding block and assignment. Thus, there are $n = k2^{\lceil \frac{N}{k} \rceil}$ different identifiers.

Let p_i be the i-th prime number. In order to compute an identifier, we initialize it to p_b, where b is the index of the identifier's corresponding block. Then, we run through all of the $M = \mathcal{O}(N)$ clauses and do the following: suppose we process the i-th clause and let $0 \le j \le 3$ be the number of variables of the identifier's corresponding assignment that satisfy the clause. We update the identifier by multiplying it with p_{k+i}^{j}.

Since each variable appears only in a constant number of clauses, each identifier is a product of $\mathcal{O}(\frac{N}{k})$ numbers. The prime number theorem guarantees $\mathcal{O}(\log N)$ bits to represent each factor, which means the identifiers have $\mathcal{O}(\frac{N}{k} \log N)$ bits. Using the fact that $n = k2^{\lceil \frac{N}{k} \rceil}$, each identifier is represented by $\mathcal{O}\left(\log n(\log k + \log\log n)\right)$ bits.

These n identifiers, along with the target $P = \prod_{i=1}^{k+M} p_i$ (recall that p_i is the i-th prime number), form a k-sized Subset Product instance. This preprocessing step costs $\mathcal{O}(2^{\frac{N}{k}})$ time, ignoring polynomial terms, which is more efficient than $\mathcal{O}(2^{s_3 N})$.

Due to the unique prime factorization, a solution to the k-sized Subset Product corresponds to a solution in f and vice-versa. If the running time of the k-sized Subset Product was $\mathcal{O}(n^{\frac{s_3 k}{101}})$ then we could solve the above instance in $\mathcal{O}((k2^{\frac{N}{k}})^{\frac{s_3 k}{101}})$ time.

Since $k = \frac{n}{2^{\lceil \frac{N}{k} \rceil}}$ and $k < n^{0.99}$, it follows that $\frac{n}{2^{\lceil \frac{N}{k} \rceil}} < n^{0.99} \implies n^{0.99} < 2^{99 \lceil \frac{N}{k} \rceil}$. But $k < n^{0.99}$, which means $k < 2^{99 \lceil \frac{N}{k} \rceil}$.

Thus the previous running time becomes $\mathcal{O}(2^{\frac{100}{101} s_3 N})$. Both the preprocessing step and the solution of the k-sized Subset Product can be achieved in time $\mathcal{O}(2^{\delta N})$, where $\delta < s_3$. However, this would violate Proposition 16. ◀

Using the above, we are ready to prove our (matching) lower bound, conditional on ETH.

▶ **Theorem 18.** *Under ETH, there is no PTAS for WLCS with running time $|I|^{\mathcal{O}(\frac{1}{\epsilon})}$, where $|I|$ is the input size in bits.*

Proof. Suppose that such an algorithm $A(I, \epsilon)$ existed. Let $R()$ be the polynomial time reduction from k-sized Subset Product to WLCS given in the proof of Theorem 14. Then, there is a solution to k-sized Subset Product iff there is a solution to WLCS of size $k + 1$, or, equivalently, iff the optimal solution to WLCS is at least $k + 1$.

Using the hypothetical $A(I, \epsilon)$ with an appropriate value of ϵ, we solve k-sized Subset Product more efficiently than possible, thus reaching a contradiction.

Consider the following algorithm for k-sized Subset Product, where there are $|L|$ numbers in the input, each having $\mathcal{O}\left(\log|L|(\log k + \log\log|L|)\right)$ bits and $k < |L|^{0.99}$. Given an instance (L, k, P), we define the instance for the WLCS to be $I = R(L, k, P)$. We run $A(I, \frac{1}{2(k+1)})$ and if the output is at least $k + 1$ we return that (L, k, P) is satisfied, otherwise we return that it cannot be satisfied.

Note that if k-sized Subset Product is solvable, then $OPT(I) \ge k + 1$, and the value output by A is at least $(1 - \frac{1}{2(k+1)})(k+1) = k + \frac{1}{2} > k$. Thus, the value output by A is at least $k + 1$. On the other hand, if k-sized Subset Product is not solvable, then $OPT(I) < k + 1$, and obviously the value output by A is at most k.

Thus we found an algorithm for k-sized Subset Product whose running time is $|I|^{\mathcal{O}(k)}$. Since I is obtained by a polynomial time reduction, its size is bounded by a polynomial in $|(L, k, P)|$. Therefore, the above running time becomes $|(L, k, P)|^{\mathcal{O}(k)}$. Under our assumptions, this becomes $|L|^{\mathcal{O}(k)}$, which is not feasible under ETH, due to Lemma 17. ◀

7 Conclusion

In this paper we prove NP-Completeness for the WLCS decision problem, and give a $PTAS$ along with a matching conditional lower bound for the optimization problem. In the most usual setting, where the alphabet size is constant, the above $PTAS$ is in fact an $EPTAS$, and it is known that no $FPTAS$ can exist unless $P = NP$. In the full version of the paper [29], we give a transformation such that algorithms for the WLCS problem can be applied for the (a_1, a_2)-WLCS problem.

In proving that WLCS does not admit any $EPTAS$, we proved that it is $W[1] - hard$. It may be interesting to determine the exact complexity of WLCS in the $W - hierarchy$.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.

2 Amihood Amir, Eran Chencinski, Costas S. Iliopoulos, Tsvi Kopelowitz, and Hui Zhang. Property matching and weighted matching. *Theoretical Computer Science*, 395(2-3):298–310, 2008. doi:10.1016/j.tcs.2008.01.006.

3 Amihood Amir, Zvi Gotthilf, and B. Riva Shalom. Weighted LCS. *Journal of Discrete Algorithms*, 8(3):273–281, 2010. doi:10.1016/j.jda.2010.02.001.

4 Amihood Amir, Zvi Gotthilf, and B. Riva Shalom. Weighted shortest common supersequence. In *String Processing and Information Retrieval, 18th International Symposium, SPIRE 2011, Pisa, Italy, October 17-21, 2011. Proceedings*, pages 44–54, 2011. doi:10.1007/978-3-642-24583-1_6.

5 Amihood Amir, Tzvika Hartman, Oren Kapah, B. Riva Shalom, and Dekel Tsur. Generalized LCS. *Theoretical Computer Science*, 409(3):438–449, 2008. doi:10.1016/j.tcs.2008.08.037.

6 Amihood Amir, Costas S. Iliopoulos, Oren Kapah, and Ely Porat. Approximate matching in weighted sequences. In *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, pages 365–376, 2006. doi:10.1007/11780441_33.

7 Pavlos Antoniou, Costas S. Iliopoulos, Laurent Mouchard, and Solon P. Pissis. Algorithms for mapping short degenerate and weighted sequences to a reference genome. *International Journal of Computational Biology and Drug Design*, 2(4):385–397, 2009. doi:10.1504/IJCBDD.2009.030768.

8 Alberto Apostolico, Gad M. Landau, and Steven Skiena. Matching for run-length encoded strings. *Journal of Complexity*, 15(1):4–16, 1999. doi:10.1006/jcom.1998.0493.

9 Carl Barton, Costas S. Iliopoulos, and Solon P. Pissis. Optimal computation of all tandem repeats in a weighted sequence. *Algorithms for Molecular Biology*, 9:21, 2014. doi:10.1186/s13015-014-0021-5.

10 Carl Barton, Tomasz Kociumaka, Chang Liu, Solon P. Pissis, and Jakub Radoszewski. Indexing weighted sequences: Neat and efficient. *Information and Computation*, 270, 2020. doi:10.1016/j.ic.2019.104462.

11 Carl Barton and Solon P. Pissis. Crochemore's partitioning on weighted strings and applications. *Algorithmica*, 80(2):496–514, 2018. doi:10.1007/s00453-016-0266-0.

12 Jon Louis Bentley, Dorothea Haken, and James B. Saxe. A general method for solving divide-and-conquer recurrences. *SIGACT News*, 12(3):36–44, September 1980. doi:10.1145/1008861.1008865.

13 Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *Seventh International Symposium on String Processing and Information Retrieval, SPIRE 2000, A Coruña, Spain, September 27-29, 2000*, pages 39–48, 2000. doi:10.1109/SPIRE.2000.878178.

14 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1216–1235, 2018. doi:10.1137/1.9781611975031.79.

15 Marco Cesati. Perfect code is W[1]-complete. *Information Processing Letters*, 81(3):163–168, 2002. doi:10.1016/S0020-0190(01)00207-1.

16 Panagiotis Charalampopoulos, Costas S. Iliopoulos, Solon P. Pissis, and Jakub Radoszewski. On-line weighted pattern matching. *Information and Computation*, 266:49–59, 2019. doi:10.1016/j.ic.2019.01.001.

17 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, and Wiktor Zuba. Weighted shortest common supersequence problem revisited. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7-9, 2019, Proceedings*, volume 11811 of *Lecture Notes in Computer Science*, pages 221–238. Springer, 2019. doi:10.1007/978-3-030-32686-9_16.

18 Manolis Christodoulakis, Costas S. Iliopoulos, Laurent Mouchard, Katerina Perdikuri, Athanasios K. Tsakalidis, and Kostas Tsichlas. Computation of repetitions and regularities of biologically weighted sequences. *Journal of Computational Biology*, 13(6):1214–1231, 2006. doi:10.1089/cmb.2006.13.1214.

19 Marek Cygan, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Polynomial-time approximation algorithms for weighted LCS problem. *Discrete Applied Mathematics*, 204:38–48, 2016. doi:10.1016/j.dam.2015.11.011.

20 Michael L. Fredman and Dan E. Willard. BLASTING through the information theoretic barrier with FUSION TREES. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 1–7, 1990. doi:10.1145/100216.100217.

21 Martin Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009. doi:10.1137/070711761.

22 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

23 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

24 David Harvey and Joris Van Der Hoeven. Integer multiplication in time O(n log n). working paper or preprint, March 2019. URL: https://hal.archives-ouvertes.fr/hal-02070778.

25 Daniel S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975. doi:10.1145/360825.360861.

26 Costas S. Iliopoulos, Christos Makris, Yannis Panagis, Katerina Perdikuri, Evangelos Theodoridis, and Athanasios K. Tsakalidis. Efficient algorithms for handling molecular weighted sequences. In *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004), 22-27 August 2004, Toulouse, France*, pages 265–278, 2004. doi:10.1007/1-4020-8141-3_22.

27 Costas S. Iliopoulos, Mirka Miller, and Solon P. Pissis. Parallel algorithms for mapping short degenerate and weighted DNA sequences to a reference genome. *International Journal of Foundations of Computer Science*, 23(2):249–259, 2012. doi:10.1142/S0129054112400114.

28 Costas S. Iliopoulos, Katerina Perdikuri, Evangelos Theodoridis, Athanasios K. Tsakalidis, and Kostas Tsichlas. Motif extraction from weighted sequences. In *String Processing and Information Retrieval, 11th International Conference, SPIRE 2004, Padova, Italy, October 5-8, 2004, Proceedings*, pages 286–297, 2004. doi:10.1007/978-3-540-30213-1_41.

29 Evangelos Kipouridis and Kostas Tsichlas. Longest common subsequence on weighted sequences. *CoRR*, abs/1901.04068, 2019. arXiv:1901.04068.

30 Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Pattern matching and consensus problems on weighted sequences and profiles. *Theory of Computing Systems*, 63(3):506–542, 2019. doi:10.1007/s00224-018-9881-2.

31 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.

32 Gene Myers. The whole genome assembly of drosophila. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, page 753, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338635.

33 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075, 2010. doi:10.1137/1.9781611973075.86.

34 Jakub Radoszewski and Tatiana Starikovskaya. Streaming k-mismatch with error correcting and applications. *Information and Computation*, 271:104513, 2020. doi:10.1016/j.ic.2019.104513.

35 Arnold Schönhage and Volker Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971. doi:10.1007/BF02242355.

36 Julie Thompson, Desmond G. Higgins, and Toby J. Gibson. W: Clustal. improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic acids research*, 22:4673–80, December 1994. doi:10.1093/nar/22.22.4673.

37 J. Craig Venter. Sequencing the human genome. In *RECOMB*, pages 309–309. ACM, 2002.

38 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

39 Hui Zhang, Qing Guo, Jing Fan, and Costas S. Iliopoulos. Loose and strict repeats in weighted sequences of proteins. *Protein and Peptide Letters*, 17(9):1136–1142, 2010.

40 Hui Zhang, Qing Guo, and Costas S. Iliopoulos. String matching with swaps in a weighted sequence. In *CIS*, volume 3314 of *Lecture Notes in Computer Science*, pages 698–704. Springer, 2004.

41 Hui Zhang, Qing Guo, and Costas S. Iliopoulos. An algorithmic framework for motif discovery problems in weighted sequences. In *Algorithms and Complexity, 7th International Conference, CIAC 2010, Rome, Italy, May 26-28, 2010. Proceedings*, pages 335–346, 2010. doi:10.1007/978-3-642-13073-1_30.

42 Hui Zhang, Qing Guo, and Costas S. Iliopoulos. Varieties of regularities in weighted sequences. In *AAIM*, volume 6124 of *Lecture Notes in Computer Science*, pages 271–280. Springer, 2010.

43 Hui Zhang, Qing Guo, and Costas S. Iliopoulos. Locating tandem repeats in weighted sequences in proteins. *BMC Bioinformatics*, 14(S-8):S2, 2013.