Weighted LCS
(Extended Abstract)

Amihood Amir[1,⋆], Zvi Gotthilf[2], and B. Riva Shalom[3]

[1] Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel
and Department of Computer Science, Johns Hopkins University,
Baltimore, MD 21218
amir@cs.biu.ac.il
[2] Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel
gotthiz@cs.biu.ac.il
[3] Department of Software Engineering, Shenkar College, Ramat-Gan 52526, Israel
riva.shalom@gmail.com

Abstract. The Longest Common Subsequence (LCS) of two strings A and B is a well studied problem having a wide range of applications. When each symbol of the input strings is assigned a positive weight the problem becomes the *Heaviest Common Subsequence* (HCS) problem. In this paper we consider a different version of weighted LCS on *Position Weight Matrices* (PWM). The Position Weight Matrix was introduced as a tool to handle a set of sequences that are not identical, yet, have many local similarities. Such a weighted sequence is a 'statistical image' of this set where we are given the probability of every symbol's occurrence at every text location. We consider two possible definitions of LCS on PWM. For the first, we solve the weighted LCS problem of z sequences in time $O(zn^{z+1})$. For the second, we prove $\mathcal{NP}$-hardness and provide an approximation algorithm.

1 Introduction

The *Longest Common Subsequence* problem, whose first famous dynamic programming solution appeared in 1974 [14], is one of the classical problems in Computer Science. The widely known string version appears in Definition 1.

Definition 1. The String Longest Common Subsequence (LCS) Problem*:*
Input: Two strings A, B of length n over alphabet Σ.
Output: The length of the longest subsequence common to both strings.

The LCS problem has been very well studied. For a survey, see [5]. The problem is mainly motivated in measuring the similarity over the input strings. An immediate example from computational biology is measuring the commonality of two DNA molecules or proteins, which may yield functional similarity between them. The well known dynamic programming solution [7] requires a running time of

⋆ Partly supported by ISF grant 35/05.

J. Fiala, J. Kratochvíl, and M. Miller (Eds.): IWOCA 2009, LNCS 5874, pp. 36–47, 2009.

$O(n^2)$, for two input strings of length n. The LCS problem had also been investigated on more general structures such as trees and matrices [2], run-length encoded strings [4], and more.

Another structure, useful in molecular biology, is the weighted sequence. This is defined as a sequence $S = s_1, .., s_{|S|}$ where a value is associated to every s_i, $i = 1..|S|$. While comparing two weighted sequences we define a weight function, W, assigning a value to every possible match between two characters one from the first sequence and the other from the second sequence. The LCS variant for these weighted sequences aims at maximizing the weight of the common subsequence, instead of its length as hereafter defined:

Definition 2. The Heaviest Common Subsequence (HCS) Problem*:*

Input: *Two strings $A = a_1..a_n$, $B = b_1..b_n$ of length n over alphabet Σ and a weight function $W : a_i \times b_j \rightarrow N$.*

Output: *A common subsequence of length l $a_{i_1}..a_{i_l} = b_{j_1}..b_{j_l}$ maximizing the sum $\sum_{k=1}^{l} W(a_{i_k}, b_{j_k})$*

Note that in contrast to sequence alignment problem, where we have a single weight for the matching of two characters, in the HCS problem the weight of the match depends on the position of the symbols in the input sequences *as well as* on the characters themselves.

Recently, another model of weighted sequences was introduced in which, at each position of the sequence, any symbol of the alphabet can occur with a certain probability. To prevent ambiguity, we refer to such sequences as *p-weighted sequences*, though in the literature they are both named weighted sequences.

Definition 3. *([9]) A p-weighted sequence $A = a_1..a_n$ over alphabet Σ, is a sequence of sets a_i, $1 \leq i \leq n$. Every a_i is a set of pairs $(s_j, \pi_i(s_j))$, where $s_j \in \Sigma$ and $\pi_i(s_j)$ is the probability of having symbol s_j at location i.*

Formally, $a_i = \{(s_j, \pi_i(s_j)) \mid s_j \neq s_l$ for $j \neq l$, and $\sum_j \pi_i(s_j) = 1\}$.

The concept of p-weighted sequences was introduced as a tool for motif discovery and local alignment. A weighted sequence is called in the biological literature a "*Position Weight Matrix*" (PWM) [12]. A p-weighted sequence of length m is a $|\Sigma| \times m$ matrix that reports the frequency of each symbol in a finite alphabet Σ for every possible location.

The first usage of PWM sequences was for relative short sequences, for example binding sites, sequences resulting from multiple alignment etc. Iliopoulos et. al. [9] considered building very large Position Weight Matrices that correspond, for example, to complete chromosome sequences that have been obtained using a whole-genome shotgun strategy [13]. By keeping all the information the whole-genome shotgun produces, it is possible to ferret out information that has been previously undetected after being faded during the consensus step. This concept is true for other applications where local similarities are thus encoded. Therefore, the necessity of developing adequate algorithms for p-weighted sequences increases.

It is natural to extend the LCS definition to p-weighted strings as a means of measuring their similarity. However the PWM model deals with probabilities,

thus values smaller than 1 are multiplied as a subsequence is extended. The heaviest common p-weighted subsequence will always be of length 1, since every added symbol reduces the total weight. Therefore, we define a **new** but related problem named *Longest Common Weighted Subsequence*, in which the weight is allowed to decrease till a certain bound, and under this restriction the longest common subsequence is sought.

The bound is set according to the certainty level required in the application. Since we consider two p-weighted sequences, we differentiate between their probabilities by denoting π_i^A the probability of occurring at the ith location of sequence A. The formal definition appears below.

Definition 4. The Longest Common Weighted Subsequence ($LCWS$) Problem:

Input: *Two p-weighted strings A, B of length n over alphabet Σ, and a constant α, $0 < \alpha \leq 1$.*

Output: *The maximal l such that there is a common subsequence of length l, $a_{i_1}..a_{i_l} = b_{j_1}..b_{j_l}$, where $\prod_{y=1}^{l}(\pi_{i_y}^A(a_{i_y}) \cdot \pi_{j_y}^B(b_{j_y})) \geq \alpha$.*

Though the $LCWS$ problem seems natural for the position weighted matrices input, in case the probabilities of the characters of one input sequence are far from being uniformly distributed, the results may be biased and not reflect a real relation between the weighted sequences. In order to prevent this effect, and obtain informative results we suggest an additional definition to the $LCWS$ problem, Longest Common Weighted Subsequence with two thresholds, referred to as $LCWS2$. In the $LCWS2$ problem, a separate probability bound is set for each of the p-weighted sequences.

Definition 5. The Longest Common Weighted Subsequence 2 ($LCWS2$) Problem:

Input: *Two p-weighted strings A, B of length n over alphabet Σ, and constants α_1, α_2, $0 < \alpha_i \leq 1$.*

Output: *The maximal l such that there is a common subsequence of length l, $a_{i_1}..a_{i_l} = b_{j_1}..b_{j_l}$, where $\prod_{y=1}^{l} \pi_{i_y}^A(a_{i_y}) \geq \alpha_1$ AND $\prod_{y=1}^{l} \pi_{j_y}^B(b_{j_y}) \geq \alpha_2$.*

In real-world applications it is rarely the case that one needs to compare only two data instances. Rather, it is important to be able to compare multiple sequences. Consequently, we generalize the $LCWS$ problems to multiple sequences and show that our algorithm generalizes in the natural way.

This paper is organized as follows: Section 2 describes related work. The $LCWS$ problem solution and its extension appear in Section 3. We consider the $LCWS2$ problem and its hardness in Section 4. Section 5 concludes the paper and poses some open questions.

2 Related Work

Jacobson and Vo [10] solved the Heaviest Common Subsequence problem by reducing it to the Heaviest Increasing Subsequence problem (HIS). Their algorithm for the Heaviest Common Subsequence runs in $O((r+n)\log n)$ time, where

r is the number of matches between A and B and n is the length of the input sequences. For small alphabets with a uniform distribution, the time may be $O(n^2 \log n)$. Recently Li [11] gave a linear space algorithm for the HCS problem.

Regarding the p-weighted sequences, Iliopoulos et al. [8] defined the problem of longest common sub*string* of p-weighted sequences, where the common sequence is *consecutive*. They suggested solving the problem using a p-weighted generalized suffix tree, in which the longest branch common to both strings is the answer. Their problem is a special case of the LCWS problem.

Amir et. al. [1] showed some conditions where p-weighted matching problems can be reduced to ordinary pattern matching problems. In their model, the probability is fixed, and the text is p-weighted while the pattern is an ordinary string. Both these assumptions are not valid for the $LCWS$ problem.

Finally, Amir et. al. [3] have defined weighted Hamming and edit distances. Although edit distance and LCS are known to be related, our model and that of [3] are different in that they consider a p-weighted text and a regular pattern. The case of Amir et. al. [3] is the special case in our model where all probabilities of the sequences equal one.

3 Longest Common Weighted Subsequence (LCWS)

The resemblance between the HCS and LCWS problems lies in the weight demands on the common subsequence. However, there is a substantial difference between the problems. The HCS maximizes a single parameter – the weight – whereas the LCWS maximizes the length under a weight restriction.

The weight bound does force the algorithm to maximize the weight at every step, yet not as a goal but rather as a byproduct. Consider the example in Fig. 1. Let the associated weight function of the HCS be multiplying the probabilities of the symbols, as given in the third table. The HCS result will be a common subsequence of length one, obtained from matching a_1 to b_3, with weight $56/81$. Nevertheless, the LCWS for $\alpha = 1/9$ will return length 2 obtained from matching a_2 to b_1 and a_3 to b_2, which has a lower probability (weight) yet respects the threshold and yields a longer subsequence.As a consequence, a new method for solving the Longest Common Weighted Subsequence problem is required.

We present a dynamic programming algorithm for the $LCWS$ problem. We construct a two dimensional table, where the columns represent the characters of the A sequence, and the rows refer to the characters of sequence B. A character in a p-weighted sequence is a table containing all symbols of Σ and the probability of appearing at that location.

As above mentioned, the core of the LCWS problem is maximizing the LCS length under a weight restriction. Consequentially, we cannot save at every entry merely the highest probability achieved so far as it may, in the future, degrade below α and would have to be discarded. We therefore save at entry i, j, for every possible length, the highest probability of a common subsequences that can be obtained from $A[1..j]$ and $B[1..i]$. We denote the variables containing this information by $l^k_{i,j}$, where k represents the length of the common subsequence. Saving these probabilities, when some $l^k_{i,j}$ is too small, we still have the

A

$\Pi_1^A(0) = 1/9$ $\Pi_1^A(1) = 8/9$	$\Pi_2^A(0) = 2/3$ $\Pi_2^A(1) = 1/3$	$\Pi_3^A(0) = 2/3$ $\Pi_3^A(1) = 1/3$
a_1	a_2	a_3

B

b_1	b_2	b_3
$\Pi_1^B(0) = 1/2$ $\Pi_1^B(1) = 1/2$	$\Pi_2^B(0) = 2/3$ $\Pi_2^B(1) = 1/3$	$\Pi_3^B(0) = 2/9$ $\Pi_3^B(1) = 7/9$

$prob(a_i, b_j)$	a_1	a_2	a_3
b_1	4/9	1/3	1/3
b_2	8/27	4/9	4/9
b_3	56/81	7/27	7/27

Fig. 1. An example of two p-weighted sequences

information regarding $l_{i,j}^{k-1}$, which may increase its length in future steps and still exceed α in weight.

As each position in a p-weighted sequence consists of $|\Sigma|$ symbols and their probabilities, when considering the matching of a_i and b_j we compute for each symbol $\sigma \in \Sigma$ the product $\pi_i^A(\sigma)\pi_j^B(\sigma)$ and select the highest value. We denote the selected value of entry i, j as $best_{i,j}$ and save the symbol yielding this probability.

We can fill the dynamic programming table in row-major order. Computing an entry i, j implies computing the most probable common subsequence of $A[1..j]$ and $B[1..i]$ of length k, $1 \leq k \leq min\{i, j\}$. Considering $l_{i,j}^k$, the correlated subsequence can be constructed by matching the a_j and b_i, selecting their *best* symbol, and by this extending a smaller subsequence, or by matching one of b_i and a_j to a previous character from the counterpart sequence. Lemma 1 formally defines the computation required for filling an entry in the dynamic programming table.

Lemma 1

$$LCWS(B[1..i], A[1...j]) = \{l_{i,j}^k\}_{k=1}^{min\{i,j\}} = \max\{l_{i,j-1}^k, \quad l_{i-1,j}^k, \quad best_{i,j} \cdot l_{i-1,j-1}^{k-1}\}.$$

Proof: An LCWS entry contains probabilities of most probable common subsequences of length k. k must start from 1, which means that only a single element was used for the common subsequence, and is bounded by the length of the longest possible common subsequence of $A[1..j]$ and $B[1..i]$, implying it cannot exceed $\min\{i, j\}$.

Computing a certain $l_{i,j}^k = x$ we will prove the optimality of x inductively on i, j. The base case is $l_{1,1}^1$ when the common subsequence consists of a single symbol obtained by matching b_1 to a character from a_1. Obviously $best_{1,1}$, yields the proper value.

Consider now $l_{i,j}^k$. Suppose to the contrary, that the values of $l_{i',j'}^k$, $i' < i$, or $j' < j$ are optimal, but x is not the optimal probability of a common subsequence

of length k of $A[1..j]$ and $B[1..i]$, implying there exists another common subsequence of length k with probability x' such that $x < x'$. The x' subsequence can be either obtained by a previous computed subsequence not including a match of $A[j]$ and $B[i]$, or by adding the current match to a $k-1$ common subsequence of $B[1..f]$ and $A[1..h]$. In the former case, the x' subsequence can include a match of $B[i]$, a matching of $A[j]$ or neither of them. Since $l^k_{i,j}$ maximizes the values of $l^k_{i-1,j}$, $l^k_{i,j-1}$, the assumption implies that there exists another subsequence of length k with probability x' where $max\{l^k_{i-1,j}, l^k_{i,j-1}\} < x'$ contradicting the induction hypothesis of optimal value of $l^k_{i',j'}$, $i' < i$, or $j' < j$. Note, that $l^k_{i-1,j-1}$ needs no separate discussion, as it is considered when computing both $l^k_{i-1,j}$, $l^k_{i,j-1}$.

The second possible case where the common subsequence has probability x', includes matching $A[j]$ and $B[i]$. The fact that $x < x'$ yields $best_{i,j} \cdot l^{k-1}_{i-1,j-1} < best_{i,j} \cdot l^{k-1}_{f,h}$, $f < 1, h < j$, contradicting the optimality of $l^{k-1}_{i-1,j-1}$, therefore this possibility is impossible as well. ∎

We can fill the whole table and then go over $\{l^k_{n,n}\}$ in decreasing order of k, and check whether $l^k_{n,n} \geq \alpha$. The first value satisfying the inequality, the relevant k is returned, as the length of the longest common subsequence under the α demands.

An example of a LCWS table where $\alpha = 0.002$ appears in Fig. 2.

Filling the table in this fashion implies computing every entry of the table requires $O(n + |\Sigma|)$ time for finding the *best* symbol and all $O(n)$ relevant probabilities, using Lemma 1. So the time complexity is $O(n^3 + |\Sigma|n^2)$.

	$\pi^A_1(a) = 0.5$ $\pi^A_1(b) = 0.4$ $\pi^A_1(c) = 0.1$	$\pi^A_2(a) = 0.3$ $\pi^A_2(b) = 0.2$ $\pi^A_2(c) = 0.5$	$\pi^A_3(a) = 0.1$ $\pi^A_3(b) = 0.1$ $\pi^A_3(c) = 0.8$	$\pi^A_4(a) = 0.4$ $\pi^A_4(b) = 0.3$ $\pi^A_4(c) = 0.3$	$\pi^A_5(a) = 0.3$ $\pi^A_5(b) = 0.7$ $\pi^A_5(c) = 0$
$\pi^B_1(a) = 0.2$ $\pi^B_1(b) = 0.4$ $\pi^B_1(c) = 0.4$	(best - 0.16) $l^1 \searrow$[b].16	(best - 0.2) $l^1 \searrow$[c].2	(best - 0.32) $l^1 \searrow$[c].32	(best - 0.12) $l^1 \rightarrow$[c] .32	(best - 0.28) $l^1 \rightarrow$[c] .32
$\pi^B_2(a) = 0.5$ $\pi^B_2(b) = 0.1$ $\pi^B_2(c) = 0.4$	(best - 0.25) $l^1 \searrow$ [a].25	(best - 0.2) $l^1 \rightarrow$ [a].25 $l^2 \searrow$[bc].032	(best - 0.32) $l^1 \searrow$[c].32 $l^2 \searrow$[cc].064	(best - 0.2) $l^1 \rightarrow$[c].32 $l^2 \searrow$[ca].064	(best - 0.15) $l^1 \rightarrow$[c].32 $l^2 \rightarrow$[ca].064
$\pi^B_3(a) = 0$ $\pi^B_3(b) = 0.9$ $\pi^B_3(c) = 0.1$	(best - 0.36) $l^1 \searrow$[b].36	(best - 0.18) $l^1 \rightarrow$[b].36 $l^2 \searrow$[ab]0.045	(best - 0.09) $l^1 \rightarrow$[b].36 $l^2 \downarrow$[cc].064 $l^3 \searrow$[bcb].0029	(best - 0.27) $l^1 \rightarrow$[b].36 $l^2 \rightarrow$[cc].064 $l^3 \searrow$[ccb].0173	(best - 0.63) $l^1 \searrow$[b].63 $l^2 \searrow$[cb].2016 $l^3 \searrow$[cab].0403
$\pi^B_4(a) = 0.6$ $\pi^B_4(b) = 0.1$ $\pi^B_4(c) = 0.3$	(best - 0.3) $l^1 \downarrow$[b].36	(best - 0.18) $l^1 \rightarrow$[b].36 $l^2 \searrow$[ba].0648	(best - 0.24) $l^1 \rightarrow$[b].36 $l^2 \rightarrow$[ba].0648 $l^3 \searrow$[abc].0108	(best - 0.24) $l^1 \downarrow$[b]0.36 $l^2 \searrow$[ba].0864 $l^3 \downarrow$[ccb].0173 $l^4 \searrow$ ——	(best - 0.18) $l^1 \downarrow$[b].63 $l^2 \downarrow$[cb].2016 $l^3 \downarrow$[cab].0403 $l^4 \searrow$[ccba].0031

Fig. 2. A LCWS Table

The space required is $O(n^2)$. Though each of the n^2 entries contains $O(n)$ probabilities and their origin. Nevertheless, due to Lemma 1, during the computation of $l_{i,j}^k$ we need only the cells adjacent to the current. Therefore when filling the hth row we keep only rows $h, h-1$ in the memory. As a consequence, at each step we save only $O(n)$ activated entries implying the space requirement is $O(n^2)$.

The time complexity can be improved if we note that the dependency on adjacent entries holds for each of the $l_{i,j}^k$s separately. In other words, we do not have to compute $l_{i,j}^k$ for all possible ks in the same iteration. We suggest improving the algorithm, by filling the table layer after layer. After the initialization of $l_{i,j}^1$ with $best_{i,j}$ values, at every step, we will have $l_{i,j}^k$, for a single k, computed for the entire table, and we will compute $l_{i,j}^{k+1}$, as these computation will be possible, according to Lemma 1.

At the end of iteration $k+1$, we check whether $l_{n,n}^{k+1} \geq \alpha$. In case the inequality is valid we consider $k+1$ as a possible answer, as we have just found that there exists a common subsequence of this length with a proper probability. We continue to compute $l_{i,j}^{k+2}$s and discard all $l_{i,j}^k$s, as their information is useless from now on.

If the contrary holds and $l_{n,n}^{k+1} < \alpha$ we return k as the length of the longest common weight subsequence, as $l_{n,n}^{k+1}$ contains the highest probability of a common subsequence of length $k+1$, due to Lemma 1. In case its probability is less than expected, there would be no other common subsequence of length $k+1$ or more respecting the weight demand.

Theorem 1. *The LCWS problem is solvable in $O(Ln^2)$ time and $O(n^2)$ space, where L is the length of the longest common weighted subsequence of the input.*

Proof: The Algorithm stops after an iteration in which the weight bound is not respected. Therefore the number of iterations performed is $L+1$. In each of them $l_{i,j}^k$ is computed for all n^2 entries of the table. This computation involves a constant number of operation, as detailed in Lemma 1. In a addition, $best_{i,j}$ is determined once in time $O(|\Sigma|n^2)$.

All in all we have $O((L+|\Sigma|)n^2)$ time requirements. Since in most usages of the Position Weight Matrix, $|\Sigma|$ is rather small, and actually a constant, the time complexity is converted to $O(Ln^2)$.

Regarding space, at each iteration we consider two probabilities $l_{i,j}^k$ and $l_{i,j}^{k+1}$. As the table consists of n^2 entries, we get space requirement of $O(n^2)$.

4 Longest Common Weighted Subsequence with Two Thresholds (LCWS2)

The $LCWS2$ problem, defined in Section 1, in which the probability of the common subsequence in each of the sequences, must exceed its α_i threshold cannot be solved in the same manner as the $LCWS$ is solved. This is due the difference between the problems that can be intuitively summarized by the following two observations.

Observation 1. *The LCWS problem allows its optimal solution to consider at every step increasing prefixes of the input strings.*

Proof: The dynamic programming solution has a single possible direction of enlarging the substrings to which it computes their $LCWS$, since all probabilities are associatively multiplied together. Therefore, computing $LCWS(A[1,i], B[1,j])$ depends merely on the LCS of prefixes of A and B shorter by one or zero symbols.

Observation 2. *It does not seem sufficient to consider at every step increasing prefixes of the input strings in order to obtain an optimal solution for the $LCWS2$ problem.*

Intuition: In this problem we execute two distinct probability multiplications and want to obtain the longest common subsequence satisfying the thresholds demand. Consequentially, we would like to multiply high probabilities in both sides. In case the current characters $A[i], B[j]$ *agree*, i.e., a single symbol, $\sigma \in \Sigma$, whose probability is highest for both positions, then adding this symbol as the match of the characters does not change the invariant of optimal solution so far.

However, when $A[i], B[j]$ do not *agree*, where there is a σ_1 whose probability is maximal in $A[i]$ but $\sigma_2 \neq \sigma_1$ has maximal probability in $B[j]$, it is not clear which symbol one should choose for the common subsequence. It may be more profitable to choose σ_1, even causing the B probability to decrease a lot, since later on a reversed case will occur and balance the probabilities. It, therefore, seems intuitive that local considerations do not suffice for computing the $LCWS2$ problem. This intuition is proven in the next subsection.

4.1 $LCWS2$ Is $\mathcal{NP}$-Hard

We prove that the $LCWS2$ problem is $\mathcal{NP}$-hard for unbounded alphabets. To this aim we define the $CWS2$ decision version:

Definition 6. The Common Weighted Substructure with 2 thresholds ($CWS2$)*:*
Input: *Two p-weighted strings A, B of length n over alphabet Σ, and constants L, α_1, α_2, $0 < \alpha_i \leq 1$.*
Output: *Does there exists a common weighted subsequence of length L, where $a_{i_1}..a_{i_L}$=$b_{j_1}..b_{j_L}$, where $(\prod_{y=1}^{L} \pi_{i_y}(a_{i_y}))\geq\alpha_1$ AND $(\prod_{y=1}^{L} \pi_{i_y}(b_{j_y}))\geq\alpha_2$.*

Theorem 2. *The $LCWS2$ problem is $\mathcal{NP}$-hard.*

Proof: We prove the hardness using a Turing reduction from the Partition problem.

Definition 7. The Partition *problem: [6]*
Input: *A finite set S and a "value" $v(s) \in Z^+$ for each $s \in S$.*
Output: *Is there a subset $S' \subseteq S$ such that $\sum_{s\in S'} v(s) = \sum_{s\in S-S'} v(s)$?*

Lemma 2. *Partition* $\leq_T^p$ *CWS*2.

Given set $S = s_1, s_2, ..., s_n$ of integers, we construct two weighted sequences $A = A_1..A_n$, $B = B_1..B_n$ both over alphabet of size $n+2$. In addition we need to set a pair of thresholds α_1, α_2 and L.

Observation 3. *The requirement that the product of the probabilities of the common sequence be higher than* α_i *is equivalent to demanding that the sum of the logarithm of the probabilities will be higher than* $\log \alpha_i$.

Proof: The observation is a direct result of the fact that the logarithm of a product equals the sum of logarithms. A special case is a zero probability that is converted to infinity. Note that the logarithms of probabilities are all negative numbers. We can simply invert the signs of all numbers, making them all positive, and require adding as many numbers as possible without exceeding (the inverted) $\log \alpha_i$. ∎

We are now ready to define the reduction. Given a set $S = \{s_1, ..., s_n\}$, we set alphabet of the LCWS2 problem to be $\Sigma = \{\sigma_1, ..., \sigma_{n+2}\}$. We define two p-weighted sequences, A and B, of length n. By definition 3, location i in a p-weighted sequence is the set of all pairs $(\sigma, \pi_i(\sigma))$, where $\sigma \in \Sigma$ and $\pi_i(\sigma)$ is the probability of having symbol σ at location i. We define the probabilities of the symbols of Σ in the following manner.

Let $sum = \sum_{s \in S} s$, the sum of all elements of S.

$$\pi_i^A(\sigma_j) = \begin{cases} s_i & j = i \\ x_i & j = n+1 \\ \infty & otherwise \end{cases} \qquad \pi_i^B(\sigma_j) = \begin{cases} sum - s_i & j = i \\ y_i & j = n+2 \\ \infty & otherwise \end{cases}$$

The value of x_i is such that $2^{-s_i} + 2^{-x_i} = 1$, and the value of y_i is such that $2^{s_i - sum} + 2^{-y_i} = 1$. They are necessary because in each location there is a single element with probability non-zero, thus we need to add a probability that, with it, will add up to 1.

Obviously the construction is done in polynomial time in the size of n, as $|\Sigma| = n+2$. For an example of the construction for set $S = \{6, 3, 4, 7\}$ see Fig. 3.

From the probabilities definition we get that the only possible symbols that can potentially be chosen for any weighted LCS with finite threshold are choosing σ_i of a_i with σ_i of b_i for $1 \leq i \leq n$.

We proceed with the Turing reduction. we perform up to $n/2$ iterations. In the ith iteration we set $\alpha_1 = sum/2$, $\alpha_2 = sum \cdot (i - 1/2)$, and $L = i$. We check whether there is a *CWS*2 with these parameters. If the answer is negative we increment i by one and start a new iteration. If no *CWS*2 was found after the $n/2$ iteration we terminate the search. If there is a *CWS*2 in iteration i, we declare a partition of S into sizes i and $n - i$.

Claim. A partition of S into size i and $n - i$ exists iff a *CWS*2 of length i was found on the ith iteration.

$$\text{A} = \begin{array}{|l|l|l|l|}\hline \pi_1^A(a)=6 & \pi_2^A(a)=\infty & \pi_3^A(a)=\infty & \pi_4^A(a)=\infty \\ \pi_1^A(b)=\infty & \pi_2^A(b)=3 & \pi_3^A(b)=\infty & \pi_4^A(b)=\infty \\ \pi_1^A(c)=\infty & \pi_2^A(c)=\infty & \pi_3^A(c)=4 & \pi_4^A(c)=\infty \\ \pi_1^A(d)=\infty & \pi_2^A(d)=\infty & \pi_3^A(d)=\infty & \pi_4^A(d)=7 \\ \pi_1^A(e)=x_1 & \pi_2^A(e)=x_2 & \pi_3^A(e)=x_3 & \pi_4^A(e)=x_4 \\ \pi_1^A(f)=\infty & \pi_2^A(f)=\infty & \pi_3^A(f)=\infty & \pi_4^A(f)=\infty \\ \hline \end{array}$$

$$\text{B} = \begin{array}{|l|l|l|l|}\hline \pi_1^B(a)=14 & \pi_2^B(a)=\infty & \pi_3^B(a)=\infty & \pi_4^B(a)=\infty \\ \pi_1^B(b)=\infty & \pi_2^B(b)=17 & \pi_3^B(b)=\infty & \pi_4^B(b)=\infty \\ \pi_1^B(c)=\infty & \pi_2^B(c)=\infty & \pi_3^B(c)=16 & \pi_4^B(c)=\infty \\ \pi_1^B(d)=\infty & \pi_2^B(d)=\infty & \pi_3^B(d)=\infty & \pi_4^B(d)=13 \\ \pi_1^B(e)=\infty & \pi_2^B(e)=\infty & \pi_3^B(e)=\infty & \pi_4^B(e)=\infty \\ \pi_1^B(f)=y_1 & \pi_2^B(f)=y_2 & \pi_3^B(f)=y_3 & \pi_4^B(f)=y_4 \\ \hline \end{array}$$

Fig. 3. The constructed sequences, according to the set $\{6, 3, 4, 7\}$

Proof: ($\Rightarrow$) Suppose there is a partition of S into two subsets $S_1 = \{s_{g_1}, s_{g_2}, .., s_{g_i}\}$ and S_2. As S_1 is a subset of the partition we know that the $\sum_{s_g \in S_1} s_g = sum/2$. Consequently, considering the $g_1, g_2, ...g_i$ characters of A and their corresponding symbol as the common subsequence, their log probabilities will sum up to $\alpha_1 = sum/2$. Due to the probabilities allocation in our construction, we are bound to select the same indices $g_1, ...g_i$ in the counterpart sequence B. Note that adding their new probabilities we get, $sum - s_{g_1} + sum - s_{g_2} + ... + sum - s_{g_i} = i \cdot sum - \sum_{s_g \in S_1} s_g = isum - 1/2sum = \alpha_2$. All in all, the existence of a partition of subsets $i, n-i$ in the set, implies a common subsequence of length i respecting the thresholds.

Clearly, there cannot be a longer $CWS2$, as addition of a single character to the common subsequence implies adding to the calculations of both sequences altogether sum, which result in $(i+1)sum$ while the sum of α_is is merely $i \cdot sum$.

($\Leftarrow$) We perform up to $n/2$ iterations. In the ith iteration we check whether there is a $CWS2$ of length i. In case the answer is negative we increment i by one and start a new iteration. If no $CWS2$ was found after the $n/2$ iteration we terminate the search, as the largest size of the smaller subset of the partition is $n/2$, so we have covered all relevant sizes.

Suppose we find a $CWS2$ of length i, where the common weighted subsequence is $A_{g_1}, A_{g_2}, ..., A_{g_i}$. Due to the construction, the common subsequence will appear in B at the same indices as in A. Obtaining the $CWS2$ implies that $\sum_{k=1}^{i} \pi_{g_k}^A(\sigma_{g_k}) \leq sum/2$ and due to the construction we get $s_{g_1} + ... + s_{g_i} \leq sum/2$. In addition, the occurrence of the $CWS2$ in B implies that $\sum_{k=1}^{i} \pi_{g_k}^B(\sigma_{g_k}) \leq sum(i - 1/2)$ which means that $sum - s_{g_1} + ... + sum - s_{g_i} = i \cdot sum - (s_{g_1} + ... + s_{g_i}) \leq sum(i - 1/2)$. Since we have just claimed that the sum of the chosen numbers from set S are less than or equal to $sum/2$, subtracting it from $i \cdot sum$ we get a result greater or equal to $sum(i - 1/2)$, contradicting the requirement of not exceeding α_2. Hence, it must be the case that $s_{g_1} + ... + s_{g_i} = sum/2$. Thus, these numbers form a subset of the partition problem. ∎

The above lemma concludes the proof of the theorem. ∎

4.2 Approximation Algorithm

Having proved that the $LCWS2$ problem for unbounded Σ is $\mathcal{NP}$-hard, we provide an approximation algorithm $LCWS2_A$.

The approximation algorithm considers each symbol $\sigma \in \Sigma$ separately. For a fixed $\sigma \in \Sigma$, let $i_1, ..., i_k$ be the indices of the longest possible sequence of σ's in A such that $\Pi_{\ell=1}^{k} \pi_{i_\ell}^{A}(\sigma) \geq \alpha_1$, and let $j_1, ..., j_m$ be the indices of the longest possible sequence of σ's in B such that $\Pi_{\ell=1}^{m} \pi_{j_\ell}^{B}(\sigma) \geq \alpha_2$. Take the minimum of k and m as $counter_\sigma$.

Choose the symbol σ with the largest $counter_\sigma$ and output $\sigma^{counter_\sigma}$.

Observation 4. *The approximation algorithm requires time* $O(|\Sigma| n \log n)$.

Proof: The input is a p-weighted sequence of length n, where each character contains Σ probabilities. We therefore construct Σ lists of length at most n and we sort each list. ∎

Theorem 3. *The approximation ratio of* $LCWS2_A$ *is* $\frac{1}{|\Sigma|}$.

Proof: Suppose the optimal length of the $LCWS2$ is OPT, and that the $LCWS2_A$ algorithm returned $counter_i$. This implies that the symbol that can be repeated most frequently, without decreasing beneath the thresholds is σ_i. The optimal solution to the $LCWS2$ problem suggested by the OPT algorithm may include several symbols. Let σ_j be the most frequent symbol in the optimal solution. Note that $counter_{\sigma_j} \leq counter_{\sigma_i}$. In addition, the number of σ_j's in the optimal solution is at least $\frac{1}{|\Sigma|} OPT$. We get $\frac{1}{|\Sigma|} OPT \leq counter_{\sigma_j} \leq counter_{\sigma_i}$. ∎

5 Conclusions and Open Problems

The main contribution of this paper is in applying the Longest Common Subsequence to a new useful structure. We define the problem of Longest Common Weighted Subsequence, considering the LCS problem applied to the important structure of p-weighted sequences. We give two possible definitions to the problem. For the first, we present a simple dynamic programming algorithm that generalizes to higher dimensions. For the second we proved $\mathcal{NP}$-hardness for unbounded alphabets, and described a proper approximation algorithm. It remains unclear what is the actual complexity class of the $LCWS2$ problem over unbounded alphabet, since we used a Turing reduction for the hardness proof.

References

1. Amir, A., Chencinski, E., Iliopoulos, C.S., Kopelowitz, T., Zhang, H.: Property Matching and Weighted Matching. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 188–199. Springer, Heidelberg (2006)
2. Amir, A., Hartman, T., Kapah, O., Shalom, B.R., Tsur, D.: Generalized LCS. In: Ziviani, N., Baeza-Yates, R. (eds.) SPIRE 2007. LNCS, vol. 4726, pp. 50–61. Springer, Heidelberg (2007)

3. Amir, A., Iliopoulos, C.S., Kapah, O., Porat, E.: Approximate Matching in Weighted Sequences. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 365–376. Springer, Heidelberg (2006)
4. Apostolico, A., Landau, G.M., Skiena, S.: Matching for run-length encoded strings. Journal of Complexity 15(1), 4–16 (1999)
5. Bergroth, L., Hakonen, H., Raita, T.: A survey of longest common subsequence algorithms. In: Proc. 7th Symposium on String Processing and Information Retrieval (SPIRE), pp. 39–48 (2000)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., New York (1979)
7. Hirschberg, D.S.: A Linear space algorithm for Computing Maximal Common Subsequences. Commun. ACM 18(6), 341–343 (1975)
8. Iliopoulos, C., Makris, C., Panagis, Y., Perdikuri, K., Theodoridis, E., Tsakalidis, A.K.: Efficient Algorithms for Handling Molecular Weighted Sequences. In: IFIP TCS, pp. 265–278 (2004)
9. Iliopoulos, C.S., Mouchard, L., Pedikuri, K., Tsakalidis, A.K.: Computing the repetitions in a weighted sequence. In: Proceedings of the 2003 Prague Stringology Conference (PSC 2003), vol. 10, pp. 91–98 (2003)
10. Jacobson, G., Vo, K.P.: Heaviest Increasing/Common Subsequence Problems. In: Apostolico, A., Galil, Z., Manber, U., Crochemore, M. (eds.) CPM 1992. LNCS, vol. 644, pp. 52–66. Springer, Heidelberg (1992)
11. Li, R.: A Linear Space Algorithm for the Heaviest Common Subsequence Problem. Utilitas Mathematica 75, 13–20 (2008)
12. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res. 22, 4673–4680 (1994)
13. Venter, J.C., Celera Genomics Corporation: The Sequence of the Human Genome. Science 291, 1304–1351 (2001)
14. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM 21, 168–173 (1974)