Weighted Shortest Common Supersequence Problem Revisited

Panagiotis Charalampopoulos[1], Tomasz Kociumaka[2,3], Solon P. Pissis[4], Jakub Radoszewski[3(✉)], Wojciech Rytter[3], Juliusz Straszyński[3], Tomasz Waleń[3], and Wiktor Zuba[3]

[1] Department of Informatics, King's College London, London, UK
panagiotis.charalampopoulos@kcl.ac.uk
[2] Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
[3] Institute of Informatics, University of Warsaw, Warsaw, Poland
{kociumaka,jrad,rytter,jks,walen,w.zuba}@mimuw.edu.pl
[4] CWI, Amsterdam, The Netherlands
solon.pissis@cwi.nl

Abstract. A weighted string, also known as a position weight matrix, is a sequence of probability distributions over some alphabet. We revisit the Weighted Shortest Common Supersequence (WSCS) problem, introduced by Amir et al. [SPIRE 2011], that is, the SCS problem on weighted strings. In the WSCS problem, we are given two weighted strings W_1 and W_2 and a threshold $\frac{1}{z}$ on probability, and we are asked to compute the shortest (standard) string S such that both W_1 and W_2 match subsequences of S (not necessarily the same) with probability at least $\frac{1}{z}$. Amir et al. showed that this problem is NP-complete if the probabilities, including the threshold $\frac{1}{z}$, are represented by their logarithms (encoded in binary).

We present an algorithm that solves the WSCS problem for two weighted strings of length n over a constant-sized alphabet in $\mathcal{O}(n^2\sqrt{z}\log z)$ time. Notably, our upper bound matches known conditional lower bounds stating that the WSCS problem cannot be solved in $\mathcal{O}(n^{2-\varepsilon})$ time or in $\mathcal{O}^*(z^{0.5-\varepsilon})$ with time, where the $\mathcal{O}^*$ notation suppresses factors polynomial with respect to the instance size (with numeric values encoded in binary), unless there is a breakthrough improving upon long-standing upper bounds for fundamental NP-hard problems (CNF-SAT and Subset Sum, respectively).

We also discover a fundamental difference between the WSCS problem and the Weighted Longest Common Subsequence (WLCS) problem, introduced by Amir et al. [JDA 2010]. We show that the WLCS problem cannot be solved in $\mathcal{O}(n^{f(z)})$ time, for any function $f(z)$, unless P = NP.

Tomasz Kociumaka was supported by ISF grants no. 824/17 and 1278/16 and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (grant no. 683064).
Jakub Radoszewski and Juliusz Straszyński were supported by the "Algorithms for text processing with errors and uncertainties" project carried out within the HOMING program of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund.

N. R. Brisaboa and S. J. Puglisi (Eds.): SPIRE 2019, LNCS 11811, pp. 221–238, 2019.
https://doi.org/10.1007/978-3-030-32686-9_16

1 Introduction

Consider two strings X and Y. A common supersequence of X and Y is a string S such that X and Y are both subsequences of S. A shortest common supersequence (SCS) of X and Y is a common supersequence of X and Y of minimum length. The Shortest Common Supersequence problem (the SCS problem, in short) is to compute an SCS of X and Y. The SCS problem is a classic problem in theoretical computer science [18,23,25]. It is solvable in quadratic time using a standard dynamic-programming approach [13], which also allows computing a shortest common supersequence of any constant number of strings (rather than just two) in polynomial time. In case of an arbitrary number of input strings, the problem becomes NP-hard [23] even when the strings are binary [25].

A weighted string of length n over some alphabet Σ is a type of uncertain sequence. The uncertainty at any position of the sequence is modeled using a subset of the alphabet (instead of a single letter), with every element of this subset being associated with an occurrence probability; the probabilities are often represented in an $n \times |\Sigma|$ matrix. These kinds of data are common in various applications where: (i) imprecise data measurements are recorded; (ii) flexible sequence modeling, such as binding profiles of molecular sequences, is required; (iii) observations are private and thus sequences of observations may have artificial uncertainty introduced deliberately [2]. For instance, in computational biology they are known as position weight matrices or position probability matrices [26].

In this paper, we study the Weighted Shortest Common Supersequence problem (the WSCS problem, in short) introduced by Amir et al. [5], which is a generalization of the SCS problem for weighted strings. In the WSCS problem, we are given two weighted strings W_1 and W_2 and a probability threshold $\frac{1}{z}$, and the task is to compute the shortest (standard) string such that both W_1 and W_2 match subsequences of S (not necessarily the same) with probability at least $\frac{1}{z}$. In this work, we show the first efficient algorithm for the WSCS problem.

A related problem is the Weighted Longest Common Subsequence problem (the WLCS problem, in short). It was introduced by Amir et al. [4] and further studied in [14] and, very recently, in [20]. In the WLCS problem, we are also given two weighted strings W_1 and W_2 and a threshold $\frac{1}{z}$ on probability, but the task is to compute the longest (standard) string S such that S matches a subsequence of W_1 with probability at least $\frac{1}{z}$ and S matches a subsequence of W_2 with probability at least $\frac{1}{z}$. For standard strings S_1 and S_2, the length of their shortest common supersequence $|\mathrm{SCS}(S_1, S_2)|$ and the length of their longest common subsequence $|\mathrm{LCS}(S_1, S_2)|$ satisfy the following folklore relation:

$$|\mathrm{LCS}(S_1, S_2)| + |\mathrm{SCS}(S_1, S_2)| = |S_1| + |S_2|. \quad (1)$$

However, an analogous relation does not connect the WLCS and WSCS problems, even though both problems are NP-complete because of similar reductions, which remain valid even in the case that both weighted strings have the same length [4,5]. In this work, we discover an important difference between the two problems.

Kociumaka et al. [21] introduced a problem called Weighted Consensus, which is a special case of the WSCS problem asking whether the WSCS of two weighted strings of length n is of length n, and they showed that the Weighted Consensus problem is NP-complete yet admits an algorithm running in pseudo-polynomial time $\mathcal{O}(n + \sqrt{z} \log z)$ for constant-sized alphabets[1]. Furthermore, it was shown in [21] that the Weighted Consensus problem cannot be solved in $\mathcal{O}^*(z^{0.5-\varepsilon})$ time for any $\varepsilon > 0$ unless there is an $\mathcal{O}^*(2^{(0.5-\varepsilon)n})$-time algorithm for the Subset Sum problem. Let us recall that the Subset Sum problem, for a set of n integers, asks whether there is a subset summing up to a given integer. Moreover, the $\mathcal{O}^*(2^{n/2})$ running time for the Subset Sum problem, achieved by a classic meet-in-the-middle approach of Horowitz and Sahni [15], has not been improved yet despite much effort; see e.g. [6].

Abboud et al. [1] showed that the Longest Common Subsequence problem over constant-sized alphabets cannot be solved in $\mathcal{O}(n^{2-\varepsilon})$ time for $\varepsilon > 0$ unless the Strong Exponential Time Hypothesis [16,17,22] fails. By (1), the same conditional lower bound applies to the SCS problem, and since standard strings are a special case of weighted strings (having one letter occurring with probability equal to 1 at each position), it also applies to the WSCS problem.

The following theorem summarizes the above conditional lower bounds on the WSCS problem.

Theorem 1 (Conditional hardness of the WSCS problem; see [1,21]). *Even in the case of constant-sized alphabets, the* Weighted Shortest Common Supersequence *problem is NP-complete, and for any $\varepsilon > 0$ it cannot be solved:*

1. *in $\mathcal{O}(n^{2-\varepsilon})$ time unless the Strong Exponential Time Hypothesis fails;*
2. *in $\mathcal{O}^*(z^{0.5-\varepsilon})$ time unless there is an $\mathcal{O}^*(2^{(0.5-\varepsilon)n})$-time algorithm for the* Subset Sum *problem.*

Our Results. We give an algorithm for the WSCS problem with pseudo-polynomial running time that depends polynomially on n and z. Note that such algorithms have already been proposed for several problems on weighted strings: pattern matching [9,12,21,24], indexing [3,7,8,11], and finding regularities [10]. In contrast, we show that no such algorithm is likely to exist for the WLCS problem.

Specifically, we develop an $\mathcal{O}(n^2\sqrt{z}\log z)$-time algorithm for the WSCS problem in the case of a constant-sized alphabet[2]. This upper bound matches the conditional lower bounds of Theorem 1. We then show that unless $P = NP$, the WLCS problem cannot be solved in $\mathcal{O}(n^{f(z)})$ time for any function $f(\cdot)$.

[1] Note that in general $z \notin \mathcal{O}^*(1)$ unless z is encoded in unary.

[2] We consider the case of $|\Sigma| = \mathcal{O}(1)$ just for simplicity. For a general alphabet, our algorithm can be modified to work in $\mathcal{O}(n^2|\Sigma|\sqrt{z}\log z)$ time.

Model of Computations. We assume the word RAM model with word size $w = \Omega(\log n + \log z)$. We consider the log-probability representation of weighted sequences, that is, we assume that the non-zero probabilities in the weighted sequences and the threshold probability $\frac{1}{z}$ are all of the form $c^{\frac{p}{2^{dw}}}$, where c and d are constants and p is an integer that fits in $\mathcal{O}(1)$ machine words.

2 Preliminaries

A *weighted string* $W = W[1] \cdots W[n]$ of length $|W| = n$ over alphabet Σ is a sequence of sets of the form

$$W[i] = \{(c,\ \pi_i^{(W)}(c))\ :\ c \in \Sigma\}.$$

Here, $\pi_i^{(W)}(c)$ is the occurrence probability of the letter c at the position $i \in [1\mathinner{.\,.}n]$.[3] These values are non-negative and sum up to 1 for a given index i.

By $W[i\mathinner{.\,.}j]$ we denote the weighted *substring* $W[i] \cdots W[j]$; it is called a prefix if $i = 1$ and a suffix if $j = |W|$.

The *probability of matching* of a string S with a weighted string W, with $|S| = |W| = n$, is

$$\mathcal{P}(S, W) = \prod_{i=1}^{n} \pi_i^{(W)}(S[i]) = \prod_{i=1}^{n} \mathcal{P}(S[i] = W[i]).$$

We say that a (standard) string S *matches a weighted string* W *with probability at least* $\frac{1}{z}$, denoted by $S \approx_z W$, if $\mathcal{P}(S, W) \geq \frac{1}{z}$. We also denote

$$\mathsf{Matched}_z(W) = \{S \in \Sigma^n : \mathcal{P}(S, W) \geq \tfrac{1}{z}\}.$$

For a string S we write $W \subseteq_z S$ if $S' \approx_z W$ for some subsequence S' of S. Similarly we write $S \subseteq_z W$ if $S \approx_z W'$ for some subsequence W' of W.

Our main problem can be stated as follows.

WEIGHTED SHORTEST COMMON SUPERSEQUENCE (WSCS(W_1, W_2, z))
Input: Weighted strings W_1 and W_2 of length up to n and a threshold $\frac{1}{z}$.
Output: A shortest standard string S such that $W_1 \subseteq_z S$ and $W_2 \subseteq_z S$.

Example 2. If the alphabet is $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, then we write the weighted string as $W = [p_1, p_2, \ldots, p_n]$, where $p_i = \pi_i^{(W)}(\mathtt{a})$; in other words, p_i is the probability that the ith letter $W[i]$ is $\mathtt{a}$. For

$$W_1 = [1,\ 0.2,\ 0.5],\ W_2 = [0.2,\ 0.5,\ 1],\ \text{and}\ z = \tfrac{5}{2},$$

we have WSCS(W_1, W_2, z) = $\mathtt{baba}$ since $W_1 \subseteq_z \mathtt{b}\underline{\mathtt{aba}}$, $W_2 \subseteq_z \underline{\mathtt{ba}}\mathtt{b}\underline{\mathtt{a}}$ (the witness subsequences are underlined), and $\mathtt{baba}$ is a shortest string with this property.

[3] For any two integers $\ell \leq r$, we use $[\ell\mathinner{.\,.}r]$ to denote the integer range $\{\ell, \ldots, r\}$.

We first show a simple solution to WSCS based on the following facts.

Observation 3 (Amir et al. [3]). *Every weighted string W matches at most z standard strings with probability at least $\frac{1}{z}$, i.e., $|\mathsf{Matched}_z(W)| \le z$.*

Lemma 4. *The set $\mathsf{Matched}_z(W)$ can be computed in $\mathcal{O}(nz)$ time if $|\Sigma| = \mathcal{O}(1)$.*

Proof. If $S \in \mathsf{Matched}_z(W)$, then $S[1\mathinner{.\,.}i] \in \mathsf{Matched}_z(W[1\mathinner{.\,.}i])$ for every index i. Hence, the algorithm computes the sets $\mathsf{Matched}_z$ for subsequent prefixes of W. Each string $S \in \mathsf{Matched}_z(W[1\mathinner{.\,.}i])$ is represented as a triple (c, p, S'), where $c = S[i]$ is the last letter of S, $p = \mathcal{P}(S, W[1\mathinner{.\,.}i])$, and $S' = S[1\mathinner{.\,.}i-1]$ points to an element of $\mathsf{Matched}_z(W[1\mathinner{.\,.}i-1])$. Such a triple is represented in $\mathcal{O}(1)$ space.

Assume that $\mathsf{Matched}_z(W[1\mathinner{.\,.}i-1])$ has already been computed. Then, for every $S' = (c', p', S'') \in \mathsf{Matched}_z(W[1\mathinner{.\,.}i-1])$ and every $c \in \Sigma$, if $p := p' \cdot \pi_i^{(W)}(c) \ge \frac{1}{z}$, then the algorithm adds (c, p, S') to $\mathsf{Matched}_z(W[1\mathinner{.\,.}i])$.

By Observation 3, $|\mathsf{Matched}_z(W[1\mathinner{.\,.}i-1])| \le z$ and $|\mathsf{Matched}_z(W[1\mathinner{.\,.}i])| \le z$. Hence, the $\mathcal{O}(nz)$ time complexity follows. □

Proposition 5. *The* WSCS *problem can be solved in $\mathcal{O}(n^2 z^2)$ time if $|\Sigma| = \mathcal{O}(1)$.*

Proof. The algorithm builds $\mathsf{Matched}_z(W_1)$ and $\mathsf{Matched}_z(W_2)$ using Lemma 4. These sets have size at most z by Observation 3. The result is the shortest string in

$$\{\mathrm{SCS}(S_1, S_2) : S_1 \in \mathsf{Matched}_z(W_1),\ S_2 \in \mathsf{Matched}_z(W_2)\}.$$

Recall that the SCS of two strings can be computed in $\mathcal{O}(n^2)$ time using a standard dynamic programming algorithm [13]. □

We substantially improve upon this upper bound in Sects. 3 and 4.

2.1 Meet-in-the-Middle Technique

In the decision version of the Knapsack problem, we are given n items with weights w_i and values v_i, and we seek for a subset of items with total weight up to W and total value at least V. In the classic meet-in-the-middle solution to the Knapsack problem by Horowitz and Sahni [15], the items are divided into two sets S_1 and S_2 of sizes roughly $\frac{1}{2}n$. Initially, the total value and the total weight is computed for every subset of elements of each set S_i. This results in two sets A, B, each with $\mathcal{O}(2^{n/2})$ pairs of numbers. The algorithm needs to pick a pair from each set such that the first components of the pairs sum up to at most W and the second components sum up to at least V. This problem can be solved in linear time w.r.t. the set sizes provided that the pairs in both sets A and B are sorted by the first component.

Let us introduce a modified version this problem.

MERGE(A, B, w)
Input: Two sets A and B of points in 2 dimensions and a threshold w.
Output: Do there exist $(x_1, y_1) \in A$, $(x_2, y_2) \in B$ such that $x_1 x_2, y_1 y_2 \geq w$?

A linear-time solution to this problem is the same as for the problem in the meet-in-the-middle solution for KNAPSACK. However, for completeness we prove the following lemma (see also [21, Lemma 5.6]):

Lemma 6 (Horowitz and Sahni [15]). *The* MERGE *problem can be solved in linear time assuming that the points in A and B are sorted by the first component.*

Proof. A pair (x, y) is *irrelevant* if there is another pair (x', y') in the same set such that $x' \geq x$ and $y' \geq y$. Observe that removing an irrelevant point from A or B leads to an equivalent instance of the MERGE problem.

Since the points in A and B are sorted by the first component, a single scan through these pairs suffices to remove all irrelevant elements. Next, for each $(x, y) \in A$, the algorithm computes $(x', y') \in B$ such that $x' \geq w/x$ and additionally x' is smallest possible. As the irrelevant elements have been removed from B, this point also maximizes y' among all pairs satisfying $x' \geq w/x$. If the elements (x, y) are processed by non-decreasing values x, the values x' do not increase, and thus the points (x', y') can be computed in $\mathcal{O}(|A| + |B|)$ time in total. ◻

3 Dynamic Programming Algorithm for WSCS

Our algorithm is based on dynamic programming. We start with a less efficient procedure and then improve it in the next section. Henceforth, we only consider computing the length of the WSCS; an actual common supersequence of this length can be recovered from the dynamic programming using a standard approach (storing the parent of each state).

For a weighted string W, we introduce a data structure that stores, for every index i, the set $\{\mathcal{P}(S, W[1\mathinner{.\,.} i]) : S \in \mathsf{Matched}_z(W[1\mathinner{.\,.} i])\}$ represented as an array of size at most z (by Observation 3) with entries in the increasing order. This data structure is further denoted as $\mathit{Freq}_i(W, z)$. Moreover, for each element $p \in \mathit{Freq}_{i+1}(W, z)$ and each letter $c \in \Sigma$, a pointer to $p' = p/\pi_{i+1}^{(W)}(c)$ in $\mathit{Freq}_i(W, z)$ is stored provided that $p' \in \mathit{Freq}_i(W, z)$. A proof of the next lemma is essentially the same as of Lemma 4.

Lemma 7. *For a weighted string W of length n, the arrays $\mathit{Freq}_i(W, z)$, with $i \in [1\mathinner{.\,.} n]$, can be constructed in $\mathcal{O}(nz)$ total time if $|\Sigma| = \mathcal{O}(1)$.*

Proof. Assume that $\mathit{Freq}_i(W, z)$ is computed. For every $c \in \Sigma$, we create a list

$$L_c = \{p \cdot \pi_{i+1}^{(W)}(c) : p \in \mathit{Freq}_i(W, z),\, p \cdot \pi_{i+1}^{(W)}(c) \geq \tfrac{1}{z}\}.$$

The lists are sorted since $\mathit{Freq}_i(W,z)$ was sorted. Then $\mathit{Freq}_{i+1}(W,z)$ can be computed by merging all the lists L_c (removing duplicates). This can be done in $\mathcal{O}(z)$ time since $\sigma = \mathcal{O}(1)$. The desired pointers can be computed within the same time complexity. □

Let us extend the WSCS problem in the following way:

> $\mathrm{WSCS}'(W_1, W_2, \ell, p, q)$:
>
> **Input:** Weighted strings W_1, W_2, an integer ℓ, and probabilities p, q.
>
> **Output:** Is there a string S of length ℓ with subsequences S_1 and S_2 such that $\mathcal{P}(S_1, W_1) = p$ and $\mathcal{P}(S_2, W_2) = q$?

In the following, a *state* in the dynamic programming denotes a quadruple (i, j, ℓ, p), where $i \in [0\,..\,|W_1|]$, $j \in [0\,..\,|W_2|]$, $\ell \in [0\,..\,|W_1| + |W_2|]$, and $p \in \mathit{Freq}_i(W_1, z)$.

Observation 8. *There are* $\mathcal{O}(n^3 z)$ *states.*

In the dynamic programming, for all states (i, j, ℓ, p), we compute

$$\mathbf{DP}[i, j, \ell, p] = \max\{q \,:\, \mathrm{WSCS}'(W_1[1\,..\,i], W_2[1\,..\,j], \ell, p, q) = \mathbf{true}\}. \quad (2)$$

Let us denote $\pi_i^k(c) = \pi_i^{(W_k)}(c)$. Initially, the array **DP** is filled with zeroes, except that the values $\mathbf{DP}[0, 0, \ell, 1]$ for $\ell \in [0\,..\,|W_1| + |W_2|]$ are set to 1. In order to cover corner cases, we assume that $\pi_0^1(c) = \pi_0^2(c) = 1$ for any $c \in \Sigma$ and that $\mathbf{DP}[i, j, \ell, p] = 0$ if (i, j, ℓ, p) is not a state. The procedure Compute implementing the dynamic-programming algorithm is shown as Algorithm 1.

Algorithm 1. Compute(W_1, W_2, z)

```
for ℓ = 0 to |W1| + |W2| do
    DP[0, 0, ℓ, 1] := 1;
foreach state (i, j, ℓ, p) in lexicographic order do
    foreach c ∈ Σ do
        x := π_i^1(c); y := π_j^2(c);
        DP[i, j, ℓ, p] := max{
            DP[i, j, ℓ, p],
            DP[i − 1, j, ℓ − 1, p/x],
            y · DP[i, j − 1, ℓ − 1, p],
            y · DP[i − 1, j − 1, ℓ − 1, p/x]
        };
return min {ℓ : DP[|W1|, |W2|, ℓ, p] ≥ 1/z for some p ∈ Freq_{|W1|}(W1, z)};
```

The correctness of the algorithm is implied by the following lemma:

Lemma 9 (Correctness of Algorithm 1). *The array* **DP** *satisfies* (2)*. In particular,* $\mathsf{Compute}(W_1, W_2, z) = \mathrm{WSCS}(W_1, W_2, z)$.

Proof. The proof that **DP** satisfies (2) goes by induction on $i+j$. The base case of $i+j=0$ holds trivially. It is simple to verify the cases that $i=0$ or $j=0$. Let us henceforth assume that $i>0$ and $j>0$.

We first show that

$$\mathbf{DP}[i,j,\ell,p] \le \max\{q : \mathrm{WSCS}'(W_1[1\,..\,i], W_2[1\,..\,j], \ell, p, q) = \mathbf{true}\}.$$

The value $q = \mathbf{DP}[i,j,\ell,p]$ was derived from $\mathbf{DP}[i-1,j,\ell-1,p/x] = q$, or $\mathbf{DP}[i,j-1,\ell-1,p] = q/y$, or $\mathbf{DP}[i-1,j-1,\ell-1,p/x] = q/y$, where $x = \pi_i^1(c)$ and $y = \pi_j^2(c)$ for some $c \in \Sigma$. In the first case, by the inductive hypothesis, there exists a string T that is a solution to $\mathrm{WSCS}'(W_1[1\,..\,i{-}1], W_2[1\,..\,j], \ell{-}1, p/x, q)$. That is, T has subsequences T_1 and T_2 such that

$$\mathcal{P}(T_1, W_1[1\,..\,i-1]) = p/x \quad \text{and} \quad \mathcal{P}(T_2, W_2[1\,..\,j]) = q.$$

Then, for $S = Tc$, $S_1 = T_1c$, and $S_2 = T_2$, we indeed have

$$\mathcal{P}(S_1, W_1[1\,..\,i]) = p \quad \text{and} \quad \mathcal{P}(S_2, W_2[1\,..\,j]) = q.$$

The two remaining cases are analogous.

Let us now show that

$$\mathbf{DP}[i,j,\ell,p] \ge \max\{q : \mathrm{WSCS}'(W_1[1\,..\,i], W_2[1\,..\,j], \ell, p, q) = \mathbf{true}\}.$$

Assume a that string S is a solution to $\mathrm{WSCS}'(W_1[1\,..\,i], W_2[1\,..\,j], \ell, p, q)$. Let S_1 and S_2 be the subsequences of S such that $\mathcal{P}(S_1, W_1) = p$ and $\mathcal{P}(S_2, W_2) = q$.

Let us first consider the case that $S_1[i] = S[\ell] \neq S_2[j]$. Then $T_1 = S_1[1\,..\,i{-}1]$ and $T_2 = S_2$ are subsequences of $T = S[1\,..\,\ell-1]$. We then have

$$p' := \mathcal{P}(T_1, W_1[1\,..\,i-1]) = p/\pi_i^1(S_1[i]).$$

By the inductive hypothesis, $\mathbf{DP}[i-1,j,\ell-1,p'] \ge q$. Hence, $\mathbf{DP}[i,j,\ell,p] \ge q$ because $\mathbf{DP}[i-1,j,\ell-1,p']$ is present as the second argument of the maximum in the dynamic programming algorithm for $c = S[\ell]$.

The cases that $S_1[i] \neq S[\ell] = S_2[j]$ and that $S_1[i] = S[\ell] = S_2[j]$ rely on the values $\mathbf{DP}[i,j-1,\ell-1,p] \ge q/y$ and $\mathbf{DP}[i-1,j-1,\ell-1,p/x] \ge q/y$, respectively.

Finally, the case that $S_1[i] \neq S[\ell] \neq S_2[j]$ is reduced to one of the previous cases by changing $S[\ell]$ to $S_1[i]$ so that S is still a supersequence of S_1 and S_2 and a solution to $\mathrm{WSCS}'(W_1[1\,..\,i], W_2[1\,..\,j], \ell, p, q)$. □

Proposition 10. *The* WSCS *problem can be solved in* $\mathcal{O}(n^3 z)$ *time if* $|\Sigma| = \mathcal{O}(1)$.

Proof. The correctness follows from Lemma 9. As noted in Observation 8, the dynamic programming has $\mathcal{O}(n^3 z)$ states. The number of transitions from a single state is constant provided that $|\Sigma| = \mathcal{O}(1)$.

Before running the dynamic programming algorithm of Proposition 10, we construct the data structures $\mathit{Freq}_i(W_1, z)$ for all $i \in [1\,..\,n]$ using Lemma 7. The last dimension in the $\mathbf{DP}[i, j, \ell, p]$ array can then be stored as a position in $\mathit{Freq}_i(W_1, z)$. The pointers in the arrays Freq_i are used to follow transitions. □

4 Improvements

4.1 First Improvement: Bounds on ℓ

Our approach here is to reduce the number of states (i, j, ℓ, p) in Algorithm 1 from $\mathcal{O}(n^3 z)$ to $\mathcal{O}(n^2 z \log z)$. This is done by limiting the number of values of ℓ considered for each pair of indices i, j from $\mathcal{O}(n)$ to $\mathcal{O}(\log z)$.

For a weighted string W, we define $\mathcal{H}(W)$ as a standard string generated by taking the most probable letter at each position, breaking ties arbitrarily. The string $\mathcal{H}(W)$ is also called the *heavy* string of W. By $d_H(S, T)$ we denote the Hamming distance of strings S and T. Let us recall an observation from [21].

Observation 11 ([21, Observation 4.3]). *If $S \approx_z W$ for a string S and a weighted string W, then $d_H(S, \mathcal{H}(W)) \le \log_2 z$.*

The lemma below follows from Observation 11.

Lemma 12. *If strings S_1 and S_2 satisfy $S_1 \approx_z W_1$ and $S_2 \approx_z W_2$, then*

$$|\mathrm{SCS}(S_1, S_2) - \mathrm{SCS}(\mathcal{H}(W_1), \mathcal{H}(W_2))| \le 2 \log_2 z.$$

Proof. By Observation 11,

$$d_H(S_1, \mathcal{H}(W_1)) \le \log_2 z \quad \text{and} \quad d_H(S_2, \mathcal{H}(W_2)) \le \log_2 z.$$

Due to the relation (1) between LCS and SCS, it suffices to show the following.

Claim. Let S_1, H_1, S_2, H_2 be strings such that $|S_1| = |H_1|$ and $|S_2| = |H_2|$. If $d_H(S_1, H_1) \le d$ and $d_H(S_2, H_2) \le d$, then $|\mathrm{LCS}(S_1, S_2) - \mathrm{LCS}(H_1, H_2)| \le 2d$.

Proof. Notice that if S'_1, S'_2 are strings resulting from S_1, S_2 by removing up to d letters from each of them, then $\mathrm{LCS}(S'_1, S'_2) \ge \mathrm{LCS}(S_1, S_2) - 2d$.

We now create strings S'_k for $k = 1, 2$, by removing from S_k letters at positions i such that $S_k[i] \ne H_k[i]$. Then, according to the observation above, we have

$$\mathrm{LCS}(S'_1, S'_2) \ge \mathrm{LCS}(S_1, S_2) - 2d.$$

Any common subsequence of S'_1 and S'_2 is also a common subsequence of H_1 and H_2 since S'_1 and S'_2 are subsequences of H_1 and H_2, respectively. Consequently,

$$\mathrm{LCS}(H_1, H_2) \ge \mathrm{LCS}(S_1, S_2) - 2d.$$

In a symmetric way, we can show that $\mathrm{LCS}(S_1, S_2) \ge \mathrm{LCS}(H_1, H_2) - 2d$. This completes the proof of the claim. □

We apply the claim for $H_1 = \mathcal{H}(W_1)$, $H_2 = \mathcal{H}(W_2)$, and $d = \log_2 z$. □

Let us make the following simple observation.

Observation 13. *If $S = \mathrm{WSCS}(W_1, W_2, z)$, then $S = \mathrm{SCS}(S_1, S_2)$ for some strings S_1 and S_2 such that $W_1 \subseteq_z S_1$ and $W_2 \subseteq_z S_2$.*

Using Lemma 12, we refine the previous algorithm as shown in Algorithm 2.

Algorithm 2. Improved1(W_1, W_2, z)

In the beginning, we apply the classic $\mathcal{O}(n^2)$-time dynamic-programming solution to the standard SCS problem on $H_1 = \mathcal{H}(W_1)$ and $H_2 = \mathcal{H}(W_2)$. It computes a 2D array T such that

$$T[i, j] = \mathrm{SCS}(H_1[1 \mathinner{.\,.} i], H_2[1 \mathinner{.\,.} j]).$$

Let us denote an interval

$$L[i, j] = [T[i, j] - \lfloor 2 \log_2 z \rfloor \;\mathinner{.\,.}\; T[i, j] + \lfloor 2 \log_2 z \rfloor].$$

We run the dynamic programming algorithm Compute restricted to states (i, j, ℓ, p) with $\ell \in L[i, j]$.
Let $\mathbf{DP}'$ denote the resulting array, restricted to states satisfying $\ell \in L[i, j]$.
We return $\min\{\ell : \mathbf{DP}'[|W_1|, |W_2|, \ell, p] \geq \frac{1}{z}$ for some $p \in \mathit{Freq}_{|W_1|}(W_1, z)\}$.

Lemma 14 (Correctness of Algorithm 2). *For every state (i, j, ℓ, p), an inequality $\mathbf{DP}'[i, j, \ell, p] \leq \mathbf{DP}[i, j, \ell, p]$ holds. Moreover, if $S = \mathrm{SCS}(S_1, S_2)$, $|S| = \ell$, $\mathcal{P}(S_1, W_1[1 \mathinner{.\,.} i]) = p \geq \frac{1}{z}$ and $\mathcal{P}(S_2, W_2[1 \mathinner{.\,.} j]) = q \geq \frac{1}{z}$, then $\mathbf{DP}'[i, j, \ell, p] \geq q$. Consequently,* Improved1$(W_1, W_2, z) = \mathrm{WSCS}(W_1, W_2, z)$.

Proof. A simple induction on $i+j$ shows that the array $\mathbf{DP}'$ is lower bounded by $\mathbf{DP}$. This is because Algorithm 2 is restricted to a subset of states considered by Algorithm 1, and because $\mathbf{DP}'[i, j, \ell, p]$ is assumed to be 0 while $\mathbf{DP}[i, j, \ell, p] \geq 0$ for states (i, j, ℓ, p) ignored in Algorithm 2.

We prove the second part of the statement also by induction on $i + j$. The base cases satisfying $i = 0$ or $j = 0$ can be verified easily, so let us henceforth assume that $i > 0$ and $j > 0$.

First, consider the case that $S_1[i] = S[\ell] \neq S_2[j]$. Let $T = S[1 \mathinner{.\,.} \ell - 1]$ and $T_1 = S_1[1 \mathinner{.\,.} i - 1]$. We then have

$$p' := \mathcal{P}(T_1, W_1[1 \mathinner{.\,.} i - 1]) = p / \pi_i^1(S_1[i]).$$

Claim. If $S_1[i] = S[\ell] \neq S_2[j]$, then $T = \mathrm{SCS}(T_1, S_2)$.

Proof. Let us first show that T is a common supersequence of T_1 and S_2. Indeed, if T_1 was not a subsequence of T, then $T_1 S_1[i] = S_1$ would not be a subsequence of $T S_1[i] = S$, and if S_2 was not a subsequence of T, then it would not be a subsequence of $T S_1[i] = S$ since $S_1[i] \neq S_2[j]$.

Finally, if T_1 and S_2 had a common supersequence T' shorter than T, then $T'S_1[i]$ would be a common supersequence of S_1 and S_2 shorter than S. □

By the claim and the inductive hypothesis, $\mathbf{DP}'[i-1, j, \ell-1, p'] \geq q$. Hence, $\mathbf{DP}'[i, j, \ell, p] \geq q$ due to the presence of the second argument of the maximum in the dynamic programming algorithm for $c = S[\ell]$. Note that (i, j, ℓ, p) is a state in Algorithm 2 since $\ell \in L[i, j]$ follows from Lemma 12.

The cases that $S_1[i] \neq S[\ell] = S_2[j]$ and that $S_1[i] = S[\ell] = S_2[j]$ use the values $\mathbf{DP}'[i, j-1, \ell-1, p] \geq q/y$ and $\mathbf{DP}'[i-1, j-1, \ell-1, p/x] \geq q/y$, respectively. Finally, the case that $S_1[i] \neq S[\ell] \neq S_2[j]$ is impossible as $S = \mathrm{SCS}(S_1, S_2)$. □

Example 15. Let $W_1 = [1, 0]$, $W_2 = [0]$ (using the notation from Example 2), and $z \geq 1$. The only strings that match W_1 and W_2 are $S_1 = \mathtt{ab}$ and $S_2 = \mathtt{b}$, respectively. We have $\mathbf{DP}[2, 1, 3, 1] = 1$ which corresponds, in particular, to a solution $S = \mathtt{abb}$ which is not an SCS of S_1 and S_2. However, $\mathbf{DP}[2, 1, 2, 1] = \mathbf{DP}'[2, 1, 2, 1] = 1$ which corresponds to $S = \mathtt{ab} = \mathrm{SCS}(S_1, S_2)$.

Proposition 16. *The* WSCS *problem can be solved in* $\mathcal{O}(n^2 z \log z)$ *time if* $|\Sigma| = \mathcal{O}(1)$.

Proof. The correctness of the algorithm follows from Lemma 14. The number of states is now $\mathcal{O}(n^2 z \log z)$ and thus so is the number of considered transitions. □

4.2 Second Improvement: Meet in the Middle

The second improvement is to apply a meet-in-the-middle approach, which is possible due to following observation resembling Observation 6.6 in [21].

Observation 17. *If* $S \approx_z W$ *for a string* S *and weighted string* W *of length* n, *then there exists a position* $i \in [1 \mathinner{.\,.} n]$ *such that*

$$S[1 \mathinner{.\,.} i-1] \approx_{\sqrt{z}} W[1 \mathinner{.\,.} i-1] \quad \textit{and} \quad S[i+1 \mathinner{.\,.} n] \approx_{\sqrt{z}} W[i+1 \mathinner{.\,.} n].$$

Proof. Select i as the maximum index with $S[1 \mathinner{.\,.} i-1] \approx_{\sqrt{z}} W[1 \mathinner{.\,.} i-1]$. □

We first use dynamic programming to compute two arrays, $\overrightarrow{\mathbf{DP}}$ and $\overleftarrow{\mathbf{DP}}$. The array $\overrightarrow{\mathbf{DP}}$ contains a subset of states from $\mathbf{DP}'$; namely the ones that satisfy $p \geq \frac{1}{\sqrt{z}}$. The array $\overleftarrow{\mathbf{DP}}$ is an analogous array defined for suffixes of W_1 and W_2. Formally, we compute $\overrightarrow{\mathbf{DP}}$ for the reversals of W_1 and W_2, denoted as $\overrightarrow{\mathbf{DP}}^R$, and set $\overleftarrow{\mathbf{DP}}[i, j, \ell, p] = \overrightarrow{\mathbf{DP}}^R[|W_1| + 1 - i, |W_2| + 1 - j, \ell, p]$. Proposition 16 yields

Observation 18. *Arrays* $\overrightarrow{\mathbf{DP}}$ *and* $\overleftarrow{\mathbf{DP}}$ *can be computed in* $\mathcal{O}(n^2 \sqrt{z} \log z)$ *time.*

Henceforth, we consider only a simpler case in which there exists a solution S to $\text{WSCS}(W_1, W_2, z)$ with a decomposition $S = S_L \cdot S_R$ such that

$$W_1[1\mathinner{.\,.} i] \subseteq_{\sqrt{z}} S_L \quad \text{and} \quad W_1[i+1\mathinner{.\,.}|W_1|] \subseteq_{\sqrt{z}} S_R \tag{3}$$

holds for some $i \in [0\mathinner{.\,.}|W_1|]$.

In the pseudocode, we use the array $L[i, j]$ from the first improvement, denoted here as $\overrightarrow{L}[i, j]$, and a symmetric array $\overleftarrow{L}$ from right to left, i.e.:

$$\overleftarrow{T}[i, j] = \text{SCS}(\mathcal{H}(W_1)[i\mathinner{.\,.}|W_1|], \mathcal{H}(W_2)[j\mathinner{.\,.}|W_2|]),$$
$$\overleftarrow{L}[i, j] = [\overleftarrow{T}[i, j] - \lfloor 2\log_2 z\rfloor \mathinner{.\,.} \overleftarrow{T}[i, j] + \lfloor 2\log_2 z\rfloor].$$

Algorithm 3 is applied for every $i \in [0\mathinner{.\,.}|W_1|]$ and $j \in [0\mathinner{.\,.}|W_2|]$.

Algorithm 3. $\mathsf{Improved2}(W_1, W_2, z, i, j)$

> $\mathit{res} := \infty$;
> **foreach** $\ell_L \in \overrightarrow{L}[i, j]$, $\ell_R \in \overleftarrow{L}[i+1, j+1]$ **do**
> $A := \{(p, q) : \overrightarrow{\mathbf{DP}}[i, j, \ell_L, p] = q\}$;
> $B := \{(p, q) : \overleftarrow{\mathbf{DP}}[i+1, j+1, \ell_R, p] = q\}$;
>   **if** $\textsc{Merge}(A, B, z)$ **then**
> $\mathit{res} := \min(\mathit{res}, \ell_L + \ell_R)$;
> **return** res;

Lemma 19 (Correctness of Algorithm 3). *Assuming that there is a solution S to* $\text{WSCS}(W_1, W_2, z)$ *that satisfies* (3)*, we have*

$$\text{WSCS}(W_1, W_2, z) = \min_{i,j}(\mathsf{Improved2}(W_1, W_2, z, i, j)).$$

Proof. Assume that $\text{WSCS}(W_1, W_2, z)$ has a solution $S = S_L \cdot S_R$ that satisfies (3) for some $i \in [0\mathinner{.\,.}|W_1|]$ and denote $\ell_L = |S_L|$, $\ell_R = |S_R|$. Let S'_L and S'_R be subsequences of S_L and S_R such that

$$p_L := \mathcal{P}(S'_L, W_1[1\mathinner{.\,.} i]) \geq \tfrac{1}{\sqrt{z}} \quad \text{and} \quad p_R := \mathcal{P}(S'_R, W_1[i+1\mathinner{.\,.}|W_1|]) \geq \tfrac{1}{\sqrt{z}}.$$

Let S''_L and S''_R be subsequences of S_L and S_R such that

$$\mathcal{P}(S''_L, W_2[1\mathinner{.\,.} j]) = q_L \quad \text{and} \quad \mathcal{P}(S''_R, W_2[j+1\mathinner{.\,.}|W_2|]) = q_R$$

for some j and $q_L q_R \geq \frac{1}{z}$.

By Lemma 14, $\overrightarrow{\mathbf{DP}}[i, j, \ell_L, p_L] \geq q_L$ and $\overleftarrow{\mathbf{DP}}[i+1, j+1, \ell_R, p_R] \geq q_R$. Hence, the set A will contain a pair (p_L, q'_L) such that $q'_L \geq q_L$ and the set B will contain a pair (p_R, q'_R) such that $q'_R \geq q_R$. Consequently, $\textsc{Merge}(A, B, z)$ will return a positive answer.

Similarly, if $\textsc{Merge}(A, B, z)$ returns a positive answer for given i, j, ℓ_L and ℓ_R, then

$$\overrightarrow{\mathbf{DP}}[i, j, \ell_L, p_L] \geq q_L \quad \text{and} \quad \overleftarrow{\mathbf{DP}}[i+1, j+1, \ell_R, p_R] \geq q_R$$

for some $p_L p_R, q_L q_R \geq \frac{1}{z}$. By Lemma 14, this implies that

$$\text{WSCS}'(W_1[1 \mathinner{.\,.} i], W_2[1 \mathinner{.\,.} j], \ell_L, p_L, q_L)$$

and

$$\text{WSCS}'(W_1[i+1 \mathinner{.\,.} |W_1|], W_2[j+1 \mathinner{.\,.} |W_2|], \ell_R, p_R, q_R)$$

have a positive answer, so

$$\text{WSCS}'(W_1, W_2, \ell_L + \ell_R, p_L p_R, q_L q_R)$$

has a positive answer too. Due to $p_L p_R, q_L q_R \geq \frac{1}{z}$, this completes the proof. □

Proposition 20. *The* WSCS *problem can be solved in* $\mathcal{O}(n^2\sqrt{z}\log^2 z)$ *time if* $|\Sigma| = \mathcal{O}(1)$.

Proof. We use the algorithm Improved2, whose correctness follows from Lemma 19 in case (3) is satisfied. The general case of Observation 17 requires only a minor technical change to the algorithm. Namely, the computation of $\overrightarrow{\mathbf{DP}}$ then additionally includes all states (i, j, ℓ, p) such that $\ell \in \overrightarrow{L}[i, j]$, $p \geq \frac{1}{z}$, and $p = \pi_i^1(c)p'$ for some $c \in \Sigma$ and $p' \in \mathit{Freq}_{i-1}(W_1, \sqrt{z})$. Due to $|\Sigma| = \mathcal{O}(1)$, the number of such states is still $\mathcal{O}(n^2\sqrt{z}\log z)$.

For every i and j, the algorithm solves $\mathcal{O}(\log^2 z)$ instances of $\textsc{Merge}$, each of size $\mathcal{O}(\sqrt{z})$. This results in the total running time of $\mathcal{O}(n^2\sqrt{z}\log^2 z)$. □

4.3 Third Improvement: Removing One log z Factor

The final improvement is obtained by a structural transformation after which we only need to consider $\mathcal{O}(\log z)$ pairs (ℓ_L, ℓ_R).

For this to be possible, we compute prefix maxima on the ℓ-dimension of the $\overrightarrow{\mathbf{DP}}$ and $\overleftarrow{\mathbf{DP}}$ arrays in order to guarantee monotonicity. That is, if $\textsc{Merge}(A, B, z)$ returns true for ℓ_L and ℓ_R, then we make sure that it would also return true if any of these two lengths increased (within the corresponding intervals).

This lets us compute, for every $\ell_L \in \overrightarrow{L}[i, j]$ the smallest $\ell_R \in \overleftarrow{L}[i, j]$ such that $\textsc{Merge}(A, B, z)$ returns true using $\mathcal{O}(\log z)$ iterations because the sought ℓ_R may only decrease as ℓ_L increases. The pseudocode is given in Algorithm 4.

Algorithm 4. Improved3(W_1, W_2, z, i, j)

foreach *state* (i, j, ℓ, p) *of* $\overrightarrow{\mathbf{DP}}$ *in lexicographic order* **do**
  $\overrightarrow{\mathbf{DP}}[i, j, \ell, p] := \max(\overrightarrow{\mathbf{DP}}[i, j, \ell, p], \overrightarrow{\mathbf{DP}}[i, j, \ell - 1, p])$;
foreach *state* (i, j, ℓ, p) *of* $\overleftarrow{\mathbf{DP}}$ *in lexicographic order* **do**
  $\overleftarrow{\mathbf{DP}}[i, j, \ell, p] := \max(\overleftarrow{\mathbf{DP}}[i, j, \ell, p], \overleftarrow{\mathbf{DP}}[i, j, \ell - 1, p])$;
$[a \mathinner{.\,.} b] := \overrightarrow{L}[i, j]$; $[a' \mathinner{.\,.} b'] := \overleftarrow{L}[i + 1, j + 1]$;
$\ell_L := a$; $\ell_R := b' + 1$; $res := \infty$;
while $\ell_L \le b$ **and** $\ell_R \ge a'$ **do**
  $A := \{(p, q) : \overrightarrow{\mathbf{DP}}[i, j, \ell_L, p] = q\}$;
  $B := \{(p, q) : \overleftarrow{\mathbf{DP}}[i + 1, j + 1, \ell_R - 1, p] = q\}$;
  if MERGE(A, B, z) **then** ▷ ℓ_R is too large for the current ℓ_L
    $\ell_R := \ell_R - 1$;
  else ▷ ℓ_R reached the target value for the current ℓ_L
    if $\ell_R \le b'$ **then** $res := \min(res, \ell_L + \ell_R)$;
    $\ell_L := \ell_L + 1$;
return *res*;

Theorem 21. *The* WSCS *problem can be solved in* $\mathcal{O}(n^2\sqrt{z}\log z)$ *time if* $|\Sigma| = \mathcal{O}(1)$.

Proof. Let us fix indices i and j. Let us denote $Freq_i(W, z)$ by $\overrightarrow{Freq}_i(W, z)$ and introduce a symmetric array

$$\overleftarrow{Freq}_i(W, z) = \{\mathcal{P}(S, W[i \mathinner{.\,.} |W|]) : S \in \mathsf{Matched}_z(W[i \mathinner{.\,.} |W|])\}.$$

In the first loop of prefix maxima computation, we consider all $\ell \in \overrightarrow{L}[i, j]$ and $p \in \overrightarrow{Freq}_i(W_1, \sqrt{z})$, and in the second loop, all $\ell \in \overleftarrow{L}[i, j]$ and $p \in \overleftarrow{Freq}_i(W_1, \sqrt{z})$. Hence, prefix maxima take $\mathcal{O}(\sqrt{z}\log z)$ time to compute.

Each step of the while-loop in Improved3 increases ℓ_L or decreases ℓ_R. Hence, the algorithm produces only $\mathcal{O}(\log z)$ instances of MERGE, each of size $\mathcal{O}(\sqrt{z})$. The time complexity follows. □

5 Lower Bound for WLCS

Let us first define the WLCS problem as it was stated in [4,14].

WEIGHTED LONGEST COMMON SUBSEQUENCE (WLCS(W_1, W_2, z))
Input: Weighted strings W_1 and W_2 of length up to n and a threshold $\frac{1}{z}$.
Output: A longest standard string S such that $S \subseteq_z W_1$ and $S \subseteq_z W_2$.

We consider the following well-known NP-complete problem [19]:

Subset Sum
Input: A set S of positive integers and a positive integer t.
Output: Is there a subset of S whose elements sum up to t?

Theorem 22. *The* WLCS *problem cannot be solved in* $\mathcal{O}(n^{f(z)})$ *time if* P $\neq$ NP.

Proof. We show the hardness result by reducing the NP-complete Subset Sum problem to the WLCS problem with a constant value of z.

For a set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers, a positive integer t, and an additional parameter $p \in [2\,..\,n]$, we construct two weighted strings W_1 and W_2 over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, each of length n^2.

Let $q_i = \frac{s_i}{t}$. At positions $i \cdot n$, for all $i = [1\,..\,n]$, the weighted string W_1 contains letter a with probability 2^{-q_i} and b otherwise, while W_2 contains a with probability $2^{\frac{1}{p-1}(q_i-1)}$ and b otherwise. All the other positions contain letter b with probability 1. We set $z = 2$.

We assume that S contains only elements smaller than t (we can ignore the larger ones and if there is an element equal to t, then there is no need for a reduction). All the weights of a are then in the interval $(\frac{1}{2}, 1)$ since $-q_i \in (-1, 0)$ and $\frac{1}{p-1}(q_i - 1) \in (-1, 0)$. Thus, since $z = 2$, letter b originating from a position $i \cdot n$ can never occur in a subsequence of W_1 or in a subsequence of W_2. Hence, every common subsequence of W_1 and W_2 is a subsequence of $(\mathtt{b}^{n-1}\mathtt{a})^n$.

For $I \subseteq [1\,..\,n]$, we have

$$\prod_{i\in I} \pi^{(W_1)}_{i\cdot n}(\mathtt{a}) = \prod_{i\in I} 2^{-s_i/t} \geq 2^{-1} = \tfrac{1}{z} \iff \sum_{i\in I} s_i \leq t$$

and

$$\prod_{i\in I} \pi^{(W_2)}_{i\cdot n}(\mathtt{a}) = \prod_{i\in I} 2^{\frac{1}{p-1}(s_i/t-1)} \geq 2^{-1} = \tfrac{1}{z} \iff$$

$$\tfrac{1}{t(p-1)}\left(\sum_{i\in I} s_i\right) - \tfrac{|I|}{p-1} \geq -1 \iff \sum_{i\in I} s_i \geq t(1 - p + |I|).$$

If I is a solution to the instance of the Subset Sum problem, then for $p = |I|$ there is a weighted common subsequence of length $n(n-1)+p$ obtained by choosing all the letters b and the letters a that correspond to the elements of I.

Conversely, suppose that the constructed WLCS instance with a parameter $p \in [2\,..\,n]$ has a solution of length at least $n(n-1)+p$. Notice that a at position $i \cdot n$ in W_1 may be matched against a at position $i' \cdot n$ in W_2 only if $i = i'$. (Otherwise, the length of the subsequence would be at most $(n - |i - i'|)n \leq (n-1)n < n(n-1)+p$). Consequently, the solution yields a subset $I \subseteq [1\,..\,n]$ of at least p indices i such that a at position $i \cdot n$ in W_1 is matched against a at position $i \cdot n$ in W_2. By the relations above, we have (a) $|I| \geq p$, (b) $\sum_{i\in I} s_i \leq t$, and (c) $\sum_{i \in I} s_i \geq t(1 - p + |I|)$. Combining these three inequalities, we obtain $\sum_{i \in I} s_i = t$ and conclude that the SUBSET SUM instance has a solution.

Hence, the SUBSET SUM instance has a solution if and only if there exists $p \in [2\,..\,n]$ such that the constructed WLCS instance with p has a solution of length at least $n(n-1)+p$. This concludes that an $\mathcal{O}(n^{f(z)})$-time algorithm for the WLCS problem implies the existence of an $\mathcal{O}(n^{2f(2)+1}) = \mathcal{O}(n^{\mathcal{O}(1)})$-time algorithm for the SUBSET SUM problem. The latter would yield $P = NP$. □

Example 23. For $S = \{3, 7, 11, 15, 21\}$ and $t = 25 = 3 + 7 + 15$, both weighted strings W_1 and W_2 are of the form:

$$\mathtt{b}^4 * \mathtt{b}^4 * \mathtt{b}^4 * \mathtt{b}^4 * \mathtt{b}^4 *,$$

where each $*$ is equal to either a or b with different probabilities.

The probabilities of choosing a's for W_1 are equal respectively to

$$\left(2^{-\frac{3}{25}}, 2^{-\frac{7}{25}}, 2^{-\frac{11}{25}}, 2^{-\frac{15}{25}}, 2^{-\frac{21}{25}}\right),$$

while for W_2 they depend on the value of p, and are equal respectively to

$$\left(2^{-\frac{22}{25(p-1)}}, 2^{-\frac{18}{25(p-1)}}, 2^{-\frac{14}{25(p-1)}}, 2^{-\frac{10}{25(p-1)}}, 2^{-\frac{4}{25(p-1)}}\right).$$

For $p = 3$, we have: $\mathrm{WLCS}(W_1, W_2, 2) = \mathtt{b}^4\,\mathtt{a}\,\mathtt{b}^4\,\mathtt{a}\,\mathtt{b}^4\,\mathtt{b}^4\,\mathtt{a}\,\mathtt{b}^4$, which corresponds to taking the first, the second, and the fourth a. The length of this string is equal to $23 = n(n-1)+p$, and its probability of matching is $\frac{1}{2} = 2^{-\frac{22}{50}} \cdot 2^{-\frac{18}{50}} \cdot 2^{-\frac{10}{50}}$. Thus, the subset $\{3, 7, 15\}$ of S consisting of its first, second, and fourth element is a solution to the SUBSET SUM problem.

References

1. Abboud, A., Backurs, A., Williams, V.V.: Tight hardness results for LCS and other sequence similarity measures. In: Guruswami, V. (ed.) 56th IEEE Annual Symposium on Foundations of Computer Science, pp. 59–78. IEEE Computer Society (2015). https://doi.org/10.1109/FOCS.2015.14
2. Aggarwal, C.C., Yu, P.S.: A survey of uncertain data algorithms and applications. IEEE Trans. Knowl. Data Eng. **21**(5), 609–623 (2009). https://doi.org/10.1109/TKDE.2008.190
3. Amir, A., Chencinski, E., Iliopoulos, C.S., Kopelowitz, T., Zhang, H.: Property matching and weighted matching. Theor. Comput. Sci. **395**(2–3), 298–310 (2008). https://doi.org/10.1016/j.tcs.2008.01.006
4. Amir, A., Gotthilf, Z., Shalom, B.R.: Weighted LCS. J. Discrete Algorithms **8**(3), 273–281 (2010). https://doi.org/10.1016/j.jda.2010.02.001
5. Amir, A., Gotthilf, Z., Shalom, B.R.: Weighted shortest common supersequence. In: Grossi, R., Sebastiani, F., Silvestri, F. (eds.) SPIRE 2011. LNCS, vol. 7024, pp. 44–54. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24583-1_6
6. Bansal, N., Garg, S., Nederlof, J., Vyas, N.: Faster space-efficient algorithms for subset sum, k-sum, and related problems. SIAM J. Comput. **47**(5), 1755–1777 (2018). https://doi.org/10.1137/17M1158203

7. Barton, C., Kociumaka, T., Liu, C., Pissis, S.P., Radoszewski, J.: Indexing weighted sequences: neat and efficient. Inf. Comput. (2019). https://doi.org/10.1016/j.ic.2019.104462
8. Barton, C., Kociumaka, T., Pissis, S.P., Radoszewski, J.: Efficient index for weighted sequences. In: Grossi, R., Lewenstein, M. (eds.) 27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016. LIPIcs, vol. 54, pp. 4:1–4:13. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.CPM.2016.4
9. Barton, C., Liu, C., Pissis, S.P.: Linear-time computation of prefix table for weighted strings & applications. Theor. Comput. Sci. **656**, 160–172 (2016). https://doi.org/10.1016/j.tcs.2016.04.029
10. Barton, C., Pissis, S.P.: Crochemore's partitioning on weighted strings and applications. Algorithmica **80**(2), 496–514 (2018). https://doi.org/10.1007/s00453-016-0266-0
11. Charalampopoulos, P., Iliopoulos, C.S., Liu, C., Pissis, S.P.: Property suffix array with applications. In: Bender, M.A., Farach-Colton, M., Mosteiro, M.A. (eds.) LATIN 2018. LNCS, vol. 10807, pp. 290–302. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77404-6_22
12. Charalampopoulos, P., Iliopoulos, C.S., Pissis, S.P., Radoszewski, J.: On-line weighted pattern matching. Inf. Comput. **266**, 49–59 (2019). https://doi.org/10.1016/j.ic.2019.01.001
13. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press (2009). https://mitpress.mit.edu/books/introduction-algorithms-third-edition
14. Cygan, M., Kubica, M., Radoszewski, J., Rytter, W., Waleń, T.: Polynomial-time approximation algorithms for weighted LCS problem. Discrete Appl. Math. **204**, 38–48 (2016). https://doi.org/10.1016/j.dam.2015.11.011
15. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. J. ACM **21**(2), 277–292 (1974). https://doi.org/10.1145/321812.321823
16. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. **62**(2), 367–375 (2001). https://doi.org/10.1006/jcss.2000.1727
17. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**(4), 512–530 (2001). https://doi.org/10.1006/jcss.2001.1774
18. Jiang, T., Li, M.: On the approximation of shortest common supersequences and longest common subsequences. SIAM J. Comput. **24**(5), 1122–1139 (1995). https://doi.org/10.1137/S009753979223842X
19. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Symposium on the Complexity of Computer Computations. pp. 85–103. The IBM Research Symposia Series, Plenum Press, New York (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
20. Kipouridis, E., Tsichlas, K.: Longest common subsequence on weighted sequences (2019). http://arxiv.org/abs/1901.04068
21. Kociumaka, T., Pissis, S.P., Radoszewski, J.: Pattern matching and consensus problems on weighted sequences and profiles. Theory Comput. Syst. **63**(3), 506–542 (2019). https://doi.org/10.1007/s00224-018-9881-2
22. Lokshtanov, D., Marx, D., Saurabh, S.: Lower bounds based on the Exponential Time Hypothesis. Bull. EATCS **105**, 41–72 (2011). http://eatcs.org/beatcs/index.php/beatcs/article/view/92
23. Maier, D.: The complexity of some problems on subsequences and supersequences. J. ACM **25**(2), 322–336 (1978). https://doi.org/10.1145/322063.322075

24. Radoszewski, J., Starikovskaya, T.: Streaming k-mismatch with error correcting and applications. In: Bilgin, A., Marcellin, M.W., Serra-Sagristà, J., Storer, J.A. (eds.) Data Compression Conference, DCC 2017, pp. 290–299. IEEE (2017). https://doi.org/10.1109/DCC.2017.14
25. Räihä, K., Ukkonen, E.: The shortest common supersequence problem over binary alphabet is NP-complete. Theor. Comput. Sci. **16**, 187–198 (1981). https://doi.org/10.1016/0304-3975(81)90075-X
26. Stormo, G.D., Schneider, T.D., Gold, L., Ehrenfeucht, A.: Use of the 'perceptron' algorithm to distinguish translational initiation sites in E. coli. Nucl. Acids Res. **10**(9), 2997–3011 (1982). https://doi.org/10.1093/nar/10.9.2997