Theoretical Computer Science 946 (2023) 113683

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Hardness and approximation of multiple sequence alignment with column score ☆

Andrea Caucchiolo, Ferdinando Cicalese *

Department of Computer Science, University of Verona, Strada le Grazie 15, Verona, 37134, Italy

ARTICLE INFO

Article history:
Received 28 February 2022
Received in revised form 5 September 2022
Accepted 24 December 2022
Available online 2 January 2023

Keywords:
Multiple sequence alignment
Column score
NP-completeness

ABSTRACT

Multiple Sequence Alignment (MSA for short) is a well known problem in the field of computational biology. In order to evaluate the quality of a solution, many different scoring functions have been introduced, the most widely used being the Sum-of-Pairs score (SP-score). It is known that computing the best MSA under the SP-score measure is NP-hard.
In this paper, we introduce a variant of the Column score (defined in Thompson et al. 1999), which we refer to as Selective Column score: Given a symbol $a \in \Sigma$, the score of the i-th column is one if and only if all symbols of the same column are a, and otherwise zero. The a-column score of an alignment is then the number of columns made of only character a.
We show that finding the optimal MSA under the Selective Column Score is NP-hard for all alphabets of size $|\Sigma| \geq 2$, and that the associated maximization problem is poly-APX-hard. We also give an approximation algorithm that almost matches the inapproximability bound.

1. Introduction

Alignments are fundamental methods for the comparison of biological sequences. They are extensively employed for assessing similarity among sequences as well as in subprocedures of more complex operations, like the computation of approximate overlaps in the context of sequencing, or reconstruction of the most probable evolution of a set of homologous sequences of nucleotides or aminoacids.

Definition 1.1 *(Multiple alignment).* Let $s_1, \ldots, s_m$, be m strings over some alphabet Σ. Let $- \notin \Sigma$ be a gap symbol and $\Sigma' = \Sigma \cup \{-\}$. A multiple alignment $\tilde{S}$ of $s_1, \ldots, s_m$ of length ℓ is a tuple of strings $\tilde{s}_1, \ldots, \tilde{s}_m \in \Sigma'$ such that

- $|\tilde{s}_1| = |\tilde{s}_2| = \cdots |\tilde{s}_m| = \ell$,
- for each $i = 1, \ldots, m$, if we remove from $\tilde{s}_i$ all and only the gap characters we obtain the string s_i,
- there does not exist a position j where a gap occurs in all $\tilde{s}_i$, i.e., for all $j = 1, \ldots, \ell$ there exists an $i \in \{1, \ldots, m\}$, such that[1] $\tilde{s}_i[j] \neq -$.

☆ This article belongs to Section A: Algorithms, automata, complexity and games, Edited by Paul Spirakis.

* Corresponding author.
E-mail address: ferdinando.cicalese@univr.it (F. Cicalese).

[1] For a string s and integer i, we denote by $s[i]$ the ith character of s.

https://doi.org/10.1016/j.tcs.2022.12.033
0304-3975/

It is useful to visualize the alignment as a matrix, where rows are the strings $\tilde{s}_i$. Therefore, for each $j = 1, \ldots, \ell$ we refer to the sequence $\tilde{s}_1[j], \ldots, \tilde{s}_m[j]$ as the jth column of the alignment.

Many proposals exist regarding the choice of the most appropriate objective function that should be optimized when computing an alignment of a set of strings [1–4]. We will discuss some of these different measures below, in the part of the introduction dedicated to related work.

In this paper, we focus on the so called *column score* analyzed in [3], where the alignment is assessed by counting the number of columns consisting of a single character.

Definition 1.2 *(Column Score).* Let $s_1, \ldots, s_m$ be a sequence of strings from some alphabet Σ. Given an alignment $\tilde{S} = \tilde{s}_1, \ldots, \tilde{s}_m$ of length ℓ of $s_1, \ldots, s_m$, and a character $a \in \Sigma$, the *a-column score* of $\tilde{S}$, denoted $cs_a(\tilde{S})$ is the number of columns made of only character a, i.e.,

$$cs_a(\tilde{S}) = |\{j \in \{1, \ldots, \ell\} \mid \forall i = 1, \ldots, m,\ \tilde{s}_i[j] = a\}|.$$

The *column score* of $\tilde{S}$, denoted $cs(\tilde{S})$ is the number of columns made of the same character, i.e.,

$$cs(\tilde{S}) = |\cup_{a \in \Sigma} \{j \in \{1, \ldots, \ell\} \mid \forall i = 1, \ldots, m,\ \tilde{s}_i[j] = a\}| = \sum_{a \in \Sigma} cs_a(\tilde{S}).$$

We prove the NP-completeness of the following problem.

Selective Column Score Alignment (SCS-Align)	
Input:	Strings $s_1, \ldots, s_m$, over some alphabet Σ, integer $M \geq \max_i \lvert s_i \rvert$, non-negative integer κ, and a *selected* character $a \in \Sigma$.
Question:	Is there an alignment $\tilde{S} = \tilde{s}_1, \ldots, \tilde{s}_m$ of $s_1, \ldots, s_m$ such that the length of $\tilde{S}$ is $\leq M$ and the a-column score of $\tilde{S}$ is $\geq \kappa$?

In contrast with other objective functions for multiple sequence alignment, to the best of our knowledge, before our result nothing was known about the tractability of computing the best possible alignment under the (selective) a-column score.

Main result. In this paper we show that the Selective Column Score Alignment problem (and several of its variants) is NP-complete and its optimization variant (asking for the alignment of maximum a-column score) is poly-APX-hard. We also give an approximation algorithm that almost matches the inapproximability bound, up to a polylogarithmic factor.

Related work. The most commonly considered objective function for optimizing multiple sequence alignments is the so called Sum-of-Pairs score (SP-score) introduced by Carrillo and Lipman [5]. The SP score is defined as the sum over all columns of the number of pairwise differences between the characters aligned in that column. The SP-score generalizes the score based on the edit distance for pairwise alignments (a multiple sequence alignments of only two strings). An optimal pairwise alignment can be computed in quadratic time by the Needleman and Wunsh dynamic programming approach [6]. In contrast, computing an optimal multiple sequence alignment based on the SP-score is known to be NP-hard [7].

Alternative scoring measures, based on the distance to a consensus string, are investigated in Li et al. [8]. In its simplest form, given an alignment a consensus string can be obtained by selecting for each column a majority symbol. The score of the alignment is the sum of the Hamming distance between each input string and the consensus sequence. The authors of [8] investigate several variants of this approach showing that they are in general NP-hard. They also consider a sort of inverse procedure, where a median string of the input string is computed that minimizes the sum of gap-bounded edit distance of each string from the median. Once this string has been found the alignment can be constructed by optimally aligning each input string to the median. A different approach to optimizing multiple sequence alignment is proposed in [4] and it is based on a measure of information discrepancy between pairs of sequences.

All the above approaches (SP-score, distance from consensus, information discrepancy) are based on pairwise comparisons of strings. More precisely, in the SP-score case, each possible pairwise comparison of the input strings is computed and the corresponding scores are added up to obtain the final score. In the case of distance to consensus, a median string is computed and the alignment score is defined as the sum of pairwise alignment scores of the input strings to the median. The column score is a basic model of scoring functions that are obtained by evaluating the set of elements in a column, rather than composing evaluations of single pairs of elements in a column. Quoting from [3], "the SP-score is calculated such that the score increases with the number of sequences correctly aligned. It is used to determine the extent to which the programs succeed in aligning some, if not all, of the sequences in an alignment. The column score is a binary score which tests the ability of the programs to align ALL of the sequences correctly."

It is not hard to see that without the upper bound M on the alignment length, computing an alignment of maximum possible column score is equivalent to finding the longest common subsequence of the input strings, which is also known

to be an NP-hard problem [9]. However, in contrast to our SCS-Align, the Longest Common Subsequence problem is easily solvable in polynomial time when the output is restricted to subsequences made of a single given character—note that an a-column score alignment asks for a subsequence of this type. Another important issue/difference with respect to modeling column score alignment as an instance of LCS is that an alignment obtained starting from a longest common subsequence might have an uncontrollable number of gaps per row. We show that the hardness result on SCS holds for an equivalent variant of the problem where we impose a bound of 1 on the number of gaps per row. In this respect, the length constraint on the solutions of SCS-Align have some similarity with the gapped common subsequences considered in [10,11]. In these papers, however, the authors consider only the special case of two input strings and do not discuss complexity issues of the general instances.

Organization of the paper. In section 2 we prove the NP-completeness and poly-APX-hardness of our basic version of the column score, Selective Column Score Alignment, by reducing from Independent Set. We also show how the reduction can be extended to variants where there is no selected character, and the bound on the length of the alignment is substituted with a bound on the number of gaps per row of the alignment—even for the case when this bound is set to one. In section 3 we provide an $\frac{M}{\log M}$-approximation algorithm for Selective Column Score Alignment. Section 4 concludes the paper with some open questions.

2. Hardness and inapproximability of SCS-Align

In this section we establish the complexity of the SCS-Align defined in the introduction by providing a reduction from the NP-complete problem Independent Set.

The NP-hardness is consequence of Lemmas 2.2 and 2.4, while Theorem 2.5 shows that SCS-Align is not approximable within a factor of $M^{1-\epsilon}$ from the optimum.

Independent Set	
Input:	An undirected graph $G = (V, E)$, an integer k
Question:	Is there a set I of nodes of size $\geq k$ such that no two nodes in S share an edge?

The following theorem states the inapproximability of Independent Set as a direct consequence of Theorem 1.1 from [12] since Max Clique coincides with Independent Set on the complementary graph:

Theorem 2.1. *For all $\epsilon > 0$, it is NP-hard to approximate* Independent Set *to within $|V|^{1-\epsilon}$*

Given a graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, and an integer k as input for the independent set problem, we define the input for the SCS-Align problem by associating each edge with a string and each node with a triplet of positions in these strings as follows. Let $M = 3|V| + 1$ be the length of the alignment we want to create. Let $V = \{v_1, v_2, \ldots, v_n\}$ be the set of nodes of G, and $E = \{e_1, e_2, \ldots, e_m\}$ its set of edges, both listed in some (arbitrarily) fixed order. Henceforth, whenever we specify the vertices of an edge $e_i \in E$, by writing $e_i = (v_u, v_w) \in E$, we assume w.l.o.g. $u < w$. Associated to each edge $e_i = (v_u, v_w)$, we build a string of length $M - 1$

$$s_i = (000)^{u-1}010(000)^{w-u-1}100(000)^{|V|-w}.$$

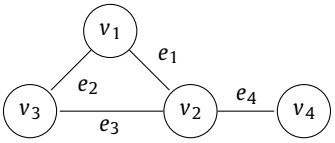
Then we create two additional strings of length M: one that leaves only one column per triplet to be able to be made of zeros, and one made of only zeros.

$$s_0 = (101)^{|V|}1$$
$$s_{m+1} = (000)^{|V|}0$$

For each string $s_0, \ldots, s_{m+1}$, and for each $j = 1, \ldots, n$, we think of the triplet of characters in position $3(j-1)+1, 3(j-1)+2, 3(j-1)+3$ as associated to the vertex v_j. Therefore, in the string s_i associated to edge $e_i = (v_u, v_w)$, we will have that each vertex is associated to a triplet of characters 000 apart from v_u which is associated to the triplet 010 and v_w to the triplet 101.

For each $i = 1, \ldots, |V|$, with *the i-th zero of s_0* we will mean the i-th occurrence of 0 in s_0 starting from the left, or equivalently, this is the 0 in the triplet of s_0 which is associated to vertex v_i.

Example 2.1. Let G be the following graph with four vertices and four edges.



Applying to G the construction described above we get the following strings:

$$s_0 = 101\ 101\ 101\ 101\ 1$$
$$s_1 = 010\ 100\ 000\ 000$$
$$s_2 = 010\ 000\ 100\ 000$$
$$s_3 = 000\ 010\ 100\ 000$$
$$s_4 = 000\ 010\ 000\ 100$$
$$s_5 = 000\ 000\ 000\ 000\ 0$$

with s_1, s_2, s_3, s_4 respectively representing the edge e_1, e_2, e_3, e_4. The spaces in the strings are meant to highlight the idea that in each string the i-th triplet of characters is associated to the i-th vertex of G.

Remark 2.1.1. Let $\tilde{S} = \tilde{s}_0, \tilde{s}_1, \ldots, \tilde{s}_{m+1}$ be an alignment of length $\ell \leq M$ of the strings $s_0, \ldots, s_{m+1}$. The following three facts are easily implied by the structure of the instance produced by our construction.

1. Since $|s_0| = |s_{m+1}| = M$ we have that it must be $\ell = M$, hence $s_0 = \tilde{s}_0$ and $s_{m+1} = \tilde{s}_{m+1}$. Moreover, since $|s_i| = M - 1$ for all $i = 1, \ldots, m$ we must have that each $\tilde{s}_i$ contains exactly one gap.
2. Following from Fact 1. and since $\tilde{s}_{m+1}$ is made of only zeroes there exist no columns with only 1, and the only columns that can contain only 0 are those corresponding to the positions of the zeroes in s_0. In other words, for the instances produced by our reduction, we have $cs_0(\tilde{S}) = cs(\tilde{S})$.
3. For each $i = 1, \ldots, m$ and each $j = 1, \ldots, M$, such that $s_0[j] = 0$, we say that $\tilde{s}_i$ *blocks* position j if $\tilde{s}_i[j] = 1$. A column j is made of only zeroes if and only if $s_0[j] = 0$ and there is no $i = 1, \ldots, m$ such that $\tilde{s}_i$ blocks position j.

For each $i = 1, \ldots, n$ let $s_i = (000)^{u-1}010(000)^{w-u-1}100(000)^{|V|-w}$ be the string associated to $e_i = (v_u, v_w)$, in the construction above. We define $s_i^U = -s_i$ to be the string obtained by adding a gap at the beginning of s_i, and $s_i^W = s_i-$ to be the string obtained by adding a gap at the end of s_i. Then, we have

$$s_i^U = -(000)^{u-1}010(000)^{w-u-1}100(000)^{|V|-w}$$
$$s_i^W = (000)^{u-1}010(000)^{w-u-1}100(000)^{|V|-w}-.$$

Example 2.2. Taking the string s_1 in Example 2.1, we have

$$s_1^U = -01\ 010\ 000\ 000\ 0$$
$$s_1^W = 010\ 100\ 000\ 000\ -$$

By direct inspection of the structure of s_i, s_0, s_i^W, s_i^U, we have the following.

Proposition 2.1.1. *Let $\tilde{S} = \tilde{s}_0, \ldots, \tilde{s}_{m+1}$ be an alignment of the strings $s_0, \ldots, s_{m+1}$ such that for each $j = 1, \ldots, m$, $\tilde{s}_j \in \{s_j^U, s_j^W\}$, and $\tilde{s}_0 = s_0$, $\tilde{s}_{m+1} = s_{m+1}$. For each $i = 1, \ldots, m$, let e_i be the edge associated with s_i and let $u, v \in [|V|]$ be such that $e_i = (v_u, v_w)$. Then, $\tilde{s}_i$ blocks exactly one position and, precisely, it blocks:*

- *the position of the w-th zero of s_0 if $\tilde{s}_i = s_i^U$;*
- *the position of the u-th zero of s_0 if $\tilde{s}_i = s_i^W$.*

Example 2.3. Let $s_0, \ldots, s_5$ be the strings in Example 2.1, and consider the alignment $\tilde{S} = \{\tilde{s}_0, \ldots, \tilde{s}_5\}$ given by $\tilde{s}_0 = s_0$, $\tilde{s}_1 = s_1^W$, $\tilde{s}_2 = s_2^U$, $\tilde{s}_3 = s_3^U$, $\tilde{s}_4 = s_4^W$, and $\tilde{s}_5 = s_5$, i.e.,

$$\tilde{s}_0 = 101\ 101\ 101\ 101\ 1$$
$$\tilde{s}_1 = 010\ 100\ 000\ 000\ -$$
$$\tilde{s}_2 = -\underline{0}1\ 000\ 0\underline{1}0\ 000\ 0$$
$$\tilde{s}_3 = -00\ 001\ 010\ 000\ 0$$
$$\tilde{s}_4 = 000\ 010\ 000\ 100\ -$$
$$\tilde{s}_5 = 000\ 000\ 000\ 000\ 0$$

As an example of Proposition 2.1.1 consider the string s_2 associated to the edge $e_2 = (v_1, v_3)$ and focus on the underlined characters of $\tilde{s}_2$. We can see that $s_2^U = -s_2$ has a 1 in the same column of the third zero of s_0 (the zero in the triplet of columns representing the vertex v_3), and is therefore blocking the position of the third zero of s_0, but no other zero of s_0, in particular, s_2 does not block the first zero, i.e., the one in the triple of positions representing v_1, which is the other vertex incident to e_2.

The next lemma proves how from a solution to the independent set problem of size at least k we can construct a solution for the SCS-Align problem of length at most M and 0-columns score at least k.

Lemma 2.2. *If there exists an independent set I for G with size at least k then there exists an alignment $\tilde{S} = \tilde{s}_0, \ldots, \tilde{s}_{m+1}$ of $s_0, \ldots, s_{m+1}$ such that the length of $\tilde{S}$ is M and there are at least k columns made only of zeroes.*

Proof. Given an independent set I for G with size at least k, we define the alignment $\tilde{S}$ as follows: $\tilde{s}_0 = s_0, \tilde{s}_{m+1} = s_{m+1}$, and for each $j = 1, \ldots, m+1$, letting $u, v \in [|V|]$ be defined by $e_j = (v_u, v_w) \in E$, we set

$$\tilde{s}_j = \begin{cases} s_j^U & \text{if } v_u \in I \\ s_j^W & \text{otherwise} \end{cases} \tag{1}$$

Then, for each $v_i \in I$, we have two cases:

- for all $e_j = (v_i, v_w) \in E$, by (1) we have $\tilde{s}_j = s_j^U$, hence by Proposition 2.1.1, $\tilde{s}_j$ does not block the position of the i-th zero of s_0.
- for all $e_j = (v_u, v_i) \in E$, by the fact that I is an independent set, we have that $v_u \notin I$. Thus, by (1) we have $\tilde{s}_j = s_j^W$, hence by Proposition 2.1.1, $\tilde{s}_j$ does not block the position of the i-th zero of s_0.

By Proposition 2.1.1, only strings associated with an edge that contains v_i can block the position of the i-th zero of s_0. Therefore, for each $v_i \in I$ the position of the i-th zero of s_0 is not blocked by any strings $\tilde{s}_j$, or equivalently, it corresponds to a column made of only zeroes. Since $|I| \geq k$, there must exist at least k columns made of zeros. □

Example 2.4. Let G be the graph in Example 2.1 and take the independent set $I = \{v_1, v_4\}$. Applying Lemma 2.2, from I we get the following alignment of the strings $s_0, \ldots, s_5$ associated to G

$\tilde{s}_0 =$ 101 101 101 101 1

$\tilde{s}_1 =$ –01 010 000 000 0

$\tilde{s}_2 =$ –01 000 010 000 0

$\tilde{s}_3 =$ 000 010 100 000 –

$\tilde{s}_4 =$ 000 010 000 100 –

$\tilde{s}_5 =$ 000 000 000 000 0

This alignment has exactly two columns made of zeros and precisely one in the first triplet representing v_1 and one in the fourth triplet representing v_4.

The next two lemmas prove the converse to Lemma 2.2, showing that from a solution to the SCS-Align problem of length at most M and 0-column score at least k we can obtain a solution for the independent set problem of size at least k. In particular, Lemma 2.3 shows that we can limit ourselves to consider alignments where for each $i = 1, \ldots, m$, the string $\tilde{s}_i$ is one of the strings s_i^U, s_i^W. Then, Lemma 2.4 shows how from such an alignment of 0-column score $\geq k$ we can construct a solution to the original independent set instance of value $\geq k$.

Lemma 2.3. *If there exists an alignment $\hat{S} = \{\hat{s}_0, \ldots, \hat{s}_{m+1}\}$ of $s_0, \ldots, s_{m+1}$ that has length M and such that $cs(\hat{S}) \geq k$, then there exists an alignment $\tilde{S} = \{\tilde{s}_0, \ldots, \tilde{s}_{m+1}\}$ of $s_0, \ldots, s_{m+1}$ that has length M such that $cs(\tilde{S}) \geq k$, and for each $i = 1, \ldots, m$ we have $\tilde{s}_i \in \{s_i^U, s_i^W\}$.*

Proof. For each edge $e_i = (v_u, v_w) \in E$ such that the associated string $\hat{s}_i \in \hat{S}$ blocks the position of the u-th zero of s_0 we set $\tilde{s}_i = s_i^W$, obtaining a string that, by Proposition 2.1.1, only blocks that position, therefore not blocking more columns than $\hat{s}_i$. Similarly, we apply the same process to all strings $\hat{s}_i$ that block the position of the w-th zero by setting $\tilde{s}_i = s_i^U$, again not blocking more columns than $\hat{s}_i$. If a string $\hat{s}_i$ blocks both the position of the u-th and w-th zero of s_0 we set $\tilde{s}_i = s_i^U$,

reducing the columns blocked by $\tilde{s}_i$ by at least one when compared to $\hat{s}_i$. It is worth noting that by construction there can not exist any string $\hat{s}_i$ that blocks neither the position of the u-th zero of s_0 nor the position of the w-th zero of s_0.

With this transformation $\tilde{S}$ does not block more columns than $\hat{S}$, therefore if $cs(\hat{S}) \geq k$ we must have $cs(\tilde{S}) \geq k$; moreover, the length of $\tilde{S}$ is equal to M since we added a gap to all strings $s_1, \ldots, s_n$ that are of length $M-1$, and for all strings $\tilde{s}_i \in \tilde{S}$ it holds $\tilde{s}_i \in \{s_i^U, s_i^W\}$ by construction. □

Example 2.5. Let $\hat{S} = \{\hat{s}_0, \hat{s}_1, \hat{s}_2, \hat{s}_3, \hat{s}_4\}$ be the following alignment of the strings $s_0, s_1, s_2, s_3, s_4, s_5$ in Example 2.1.

$\hat{s}_0 =$ 101 101 101 101 1

$\hat{s}_1 =$ 010 100 00– 000 0

$\hat{s}_2 =$ 010 –00 010 000 0

$\hat{s}_3 =$ 0–0 001 010 000 0

$\hat{s}_4 =$ 000 010 000 10– 0

$\hat{s}_5 =$ 000 000 000 000 0

It is not hard to verify that $cs(\hat{S}) = 1$. By applying the procedure described in Lemma 2.3 we get the following alignment $\tilde{S}$:

$\tilde{s}_0 =$ 101 101 101 101 1

$\tilde{s}_1 =$ 010 100 000 000 –

$\tilde{s}_2 =$ –01 000 001 000 0

$\tilde{s}_3 =$ –00 001 010 000 0

$\tilde{s}_4 =$ 000 010 000 100 –

$\tilde{s}_5 =$ 000 000 000 000 0

We can see that the alignment $\tilde{S}$ has $cs(\tilde{S}) = 1$ too and for each i, it holds that $\tilde{s_i} \in \{s_i^U, s_i^W\}$.

Lemma 2.4. *If there exists an alignment $\tilde{S}$ of $s_0, \ldots, s_{n+1}$ that has length M and such that $cs(\tilde{S}) \geq k$, then G admits an independent set I of size at least k.*

Proof. By Lemma 2.3 we can assume without loss of generality that the alignment $\tilde{S} = \{\tilde{s}_0, \tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_{n+1}\}$ satisfies $\tilde{s}_i \in \{s_i^U, s_i^W\}$ for each $i = 1 \ldots, n$. We define a set of vertexes $I \subseteq V$ by letting $v_i \in I$ if the position of the i-th zero of s_0 is not blocked by any string $\tilde{s_j} \in \tilde{S}$, or equivalently, if it coincides with a column made of zeroes. This guarantees that I has size at least k since by hypothesis $\tilde{S}$ has a column score $\geq k$.

We now show that I is an independent set. The argument is by contradiction: Let us suppose (*absurdum hypothesis*) that there exist two distinct nodes $v_i, v_j \in I$ such that $e_k = (v_i, v_j) \in E$. Let us then consider the string $\tilde{s}_k$ in the alignment $\tilde{S}$. If $\tilde{s}_k = s_k^U$ then it blocks the position of the j-th zero of s_0. Conversely, if $\tilde{s}_k = s_k^W$ then it blocks the position of the i-th zero of s_0. In either case, we have a contradiction with the hypothesis that both the positions of the i-th and the j-th zero of s_0 are not blocked. □

Example 2.6. Let $\tilde{S} = \{\tilde{s}_0, \tilde{s}_1, \tilde{s}_2, \tilde{s}_3, \tilde{s}_4, \tilde{s}_5\}$ be the following alignment of the strings $s_0, s_1, s_2, s_3, s_4, s_5$ from Example 2.1.

$\tilde{s}_0 =$ 101 101 101 101 1

$\tilde{s}_1 =$ –01 010 000 000 0

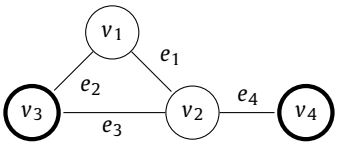
$\tilde{s}_2 =$ 010 000 100 000 –

$\tilde{s}_3 =$ 000 010 100 000 –

$\tilde{s}_4 =$ 000 010 000 100 –

$\tilde{s}_5 =$ 000 000 000 000 0

We have $cs(\hat{S}) = 2$. In particular, this column score is attained by the columns of the 3rd and the 4th zero of s_0, which are both not blocked by any other string. Applying the construction in Lemma 2.4, we set $I = \{v_3, v_4\}$.



The picture shows the graph G with the vertices in I marked in bold, from which it is easy to verify that I is indeed an independent set, and in particular $|I| = cs(\tilde{S}) = 2$.

Theorem 2.5. *For any $\epsilon > 0$, unless NP=P, there is no polynomial-time algorithm that approximates* SCS-ALIGN *within a factor $n_{\max}^{1-\epsilon}$, where $n_{\max}$ is the length of the longest string in the input instance.*

Proof. We proceed by contradiction: Let us suppose (*absurdum hypothesis*) there exist an $\epsilon > 0$ and polynomial-time algorithm $\mathcal{A}$ that approximates SCS-ALIGN within a factor of $n_{\max}^{1-\epsilon}$.

Fix a real number ϵ' and a positive integer $\hat{n}$ such that $0 < \epsilon' \leq \epsilon$ and $4n^{1-\epsilon} \leq n^{1-\epsilon'}$ for all $n \geq \hat{n}$. We are going to show that the existence of algorithm $\mathcal{A}$ implies the existence of an algorithm for INDEPENDENT SET with approximation guarantee $|V|^{1-\epsilon'}$, which is ruled out by Theorem 2.1.

Let $G = (V, E)$ be an instance of INDEPENDENT SET such that $|V| \geq \hat{n}$. By applying our reduction to G we get an instance of SCS-ALIGN consisting of the set of strings $S = \{s_0, \ldots, s_{|E|+1}\}$ where the largest string has length $n_{\max} = 3|V| + 1 \leq 4|V|$. The algorithm $\mathcal{A}$ guarantees to find a solution which is an $n_{\max}^{1-\epsilon'}$ approximation of the optimum. We define $OPT_{\text{SCS-ALIGN}}(S)$ as the maximum column score for any alignment of the strings S, and $OPT_{\text{INDSET}}(G)$ as the maximum size of any independent set of G.

Then the algorithm $\mathcal{A}$ finds an alignment $\tilde{S}$ such that $\frac{OPT_{\text{SCS-ALIGN}}(S)}{cs(\tilde{S})} \leq n_{\max}^{1-\epsilon}$. By Lemmas 2.2 and 2.4 we have that every independent set of G corresponds to an alignment of S whose column score is equal to the size of the independent set; the vice versa also holds. This implies $OPT_{\text{SCS-ALIGN}}(S) = OPT_{\text{INDSET}}(G)$. Moreover, from the column score $\tilde{S}$ output by $\mathcal{A}$ we can obtain a corresponding independent set I of G such that $cs(\tilde{S}) = |I|$.

Since $n_{\max} = 3|V| + 1$ and $|V| \geq \hat{n}$ we have,

$$\frac{OPT_{\text{INDSET}}(G)}{|I|} = \frac{OPT_{\text{SCS-ALIGN}}(S)}{cs(\tilde{S})} \leq n_{\max}^{1-\epsilon} = (3|V| + 1)^{1-\epsilon} \leq 4|V|^{1-\epsilon} \leq |V|^{1-\epsilon'},$$

meaning that with algorithm $\mathcal{A}$ we can obtain an independent set for G which is a $|V|^{1-\epsilon'}$ approximation of the optimum, which is a contradiction to the inapproximability bound of Theorem 2.1. □

Corollary 2.5.1. *For any $\epsilon > 0$, unless NP=P, there is no polynomial-time algorithm that approximates* SCS-ALIGN *within a factor $M^{1-\epsilon}$, where M is the bound on the length of the output alignment.*

Proof. By Theorem 2.5, noticing that for the SCS-ALIGN instance produced by the reduction employed in the proof, we have $M = n_{\max}$. □

2.1. Variants and extensions

In this section we show that the NP-completeness and poly-APX-hardness shown for SCS-ALIGN naturally extends to more natural variants or reformulations of the problem that are also of interest. We start by the simple observation that the results of the previous section trivially hold when the objective function is the column score rather than the a-column score with respect to a specific character a, i.e., for the following version of the problem:

BOUNDED COLUMN SCORE ALIGNMENT (BCS-ALIGN)	
Input:	Strings $s_1, \ldots, s_m$ over some alphabet Σ, an integer $M \geq \max_i \|s_i\|$, and a non-negative integer κ.
Output:	An alignment $\tilde{S} = \tilde{s}_1, \ldots, \tilde{s}_m$ of $s_1, \ldots, s_m$ of length at most M and maximum column score

In fact, due to the presence of the string s_{n+1} being made of only zeroes as observed in point 2 of Remark 2.1.1, the proof in the previous section also shows the result in the following corollary. It is enough to observe that s_{n+1} forces only columns entirely made of 0 to be accepted.

Corollary 2.5.2. *The* BOUNDED COLUMN SCORE ALIGNMENT *problem is NP-hard and if $P \neq NP$, for any $\epsilon > 0$ no polynomial time algorithm exists with approximation guarantee $M^{1-\epsilon}$.*

We can also reformulate the problem by interpreting the bound on the length of the alignment as a bound on the number of gaps. In fact, the following formulation is also easily shown to be poly-APX-hard

GAP-BOUNDED COLUMN SCORE ALIGNMENT (GAP-CS-ALIGN)	
Input:	Strings $s_1, \ldots, s_m$ over some alphabet Σ, a non-negative integer g
Output:	An alignment $\tilde{S} = \tilde{s}_1, \ldots, \tilde{s}_m$ of $s_1, \ldots, s_m$ s.t. for each $i \in [m]$, $\tilde{s}_i$ contains at most g gaps, and $cs(\tilde{S})$ is maximum

Corollary 2.5.3. *If $P \neq NP$ there exists no polynomial time algorithm for the* GAP-BOUNDED COLUMN SCORE ALIGNMENT *problem, with approximation guarantee $n_{\max}^{1-\epsilon}$, where $n_{\max} = \max_{i=1,\ldots m} |s_i|$.*

Proof. The reduction and the analysis employed for the SCS-ALIGN and its optimization variant shows that the particular case of GAP-CS-ALIGN with $g = 1$ cannot be approximated in polynomial time within a factor $n_{\max}^{1-\epsilon}$, unless $P = NP$. The simple observation needed is that in any alignment $\tilde{S} = \tilde{s}_0, \ldots, \tilde{s}_{n+1}$ of the strings $s_0, \ldots, s_{n+1}$ no gap can be present in $\tilde{s}_0$, (nor in $\tilde{s}_{n+1}$); otherwise, at least 2 gaps should be present in each $\tilde{s}_i$ $(i = 1, \ldots, n)$. Hence, we have that Remark 2.1.1 (and in particular point 1.) holds also on the basis of the bound on the number of gaps. □

3. An approximation algorithm for BCS-ALIGN

In this section we present an approximation algorithm whose guarantee "almost" matches the inapproximability bound of the previous section (Corollaries 2.5.1 and 2.5.2). More precisely, we show an $\frac{M}{\log M}$-approximation algorithm for BCS-ALIGN, when the input strings are from a bounded alphabet. Since Corollary 2.5.2 rules out any $M^{1-\epsilon}$-approximation, it means that, if $P \neq NP$ one cannot hope for more than a polylogarithmic factor improvement of our approximation guarantee.

In order to explain our approach, it is useful to introduce some more notation.

Definition 3.1 *(Sections).* Let $I = (S, M)$ be an instance of BCS-ALIGN, where $S = \{s_1, \ldots, s_m\}$ is a set of strings and M is the upper bound on the length of the alignment. Given an alignment $\tilde{S}$ of S, for each $i = 1, 2, \ldots, \frac{M}{\log M}$ we refer to the *i-th section* of $\tilde{S}$ as the set of the columns of indices $(i-1)\log M + 1, (i-1)\log M + 2, \ldots, i \log M$.[2] We define $cs(\tilde{S}, i)$ to be the number of columns in the ith section of $\tilde{S}$ that are made of the same character.

Hence, we have

$$cs(\tilde{S}) = \sum_{i=1}^{\frac{M}{\log M}} cs(\tilde{S}, i), \quad \text{and} \quad cs(\tilde{S}) \leq \frac{M}{\log M} \max_{i=1,\ldots,\frac{M}{\log M}} cs(\tilde{S}, i). \tag{2}$$

The following algorithm constructs an alignment $\tilde{S}$ such that

$$\max_{i=1,\ldots,\frac{M}{\log M}} cs(\tilde{S}, i) = \max_{\hat{S}} \max_{i=1,\ldots,\frac{M}{\log M}} cs(\hat{S}, i), \tag{3}$$

i.e., $\tilde{S}$ achieves the best possible column score achievable in a single section among all possible alignments $\hat{S}$ for S.

Note that, as shown in the following example, an alignment that satisfies (3) is not necessarily optimum.

Example 3.1. Consider an instance with the following two strings, and $M = 8$. Each section is made of 3 columns, which we visualized by adding a space between consecutive sections:

$s_0 = 011\ 100\ 1$

$s_1 = 010\ 110\ 10$

The following alignment $\tilde{S} = \{\tilde{s}_1, \tilde{s}_2\}$ achieves the maximum in (3) since $\max_i cs(\tilde{S}, i) = cs(\tilde{S}, 2) = 3$, and has column score $cs(\tilde{S}) = 4$

$\tilde{s}_0 = 0\text{–}1\ 110\ 01$

$\tilde{s}_1 = 010\ 110\ 10$

[2] For the sake of simplifying the notation, we assume that $\log M$ and $\frac{M}{\log M}$ are integers. The argument can easily be generalized by using $\lfloor \log M \rfloor$ and $\lceil \frac{M}{\log M} \rceil$ instead.

On the other hand, the alignment $\hat{S} = \{\hat{s}_1, \hat{s}_2\}$ below can be shown to be optimal (in particular, we have $cs(\hat{S}) = 5$) and it does not satisfy (3) since $\max_i cs(\hat{S}, i) = cs(\hat{S}, 1) = 2$.

$$\hat{s}_0 = 011\ 100\ 1-$$

$$\hat{s}_1 = 010\ 110\ 10$$

The algorithm (see Algorithm 1) is based on an exhaustive search procedure, that iteratively, for each section i (line 4 of the algorithm), finds the set of columns C (line 5) and a sequence of characters $c_1, \ldots, c_{|C|}$ (line 6), one for each of the chosen columns, such that it is possible to align each one of the input strings so to match on the chosen columns and the chosen characters. For this we build a string s^* made of all gaps apart from the chosen positions C where the characters are those of the chosen sequence $c_1, \ldots, c_{|C|}$. We then check whether for each input string it is possible to align it to s^* (line 8) so that all the non-gaps of s^* are exactly matched. Each such check can be done with a simple modification of the dynamic programming approach for the edit distance with the additional constraint that no additional gap can be inserted in s^*. If this happens (line 11), we have found an alignment whose column score in the ith section is equal to $|C|$. The algorithm stores the best (largest) choice of columns and characters for each section and returns the alignment that obtains the maximum over all sections.

Algorithm 1: Section-Align Algorithm.

```
1  Input: A set of strings S over the alphabet Σ and a bound M
2  Output: an alignment S̃ for S of size ≤ M that achieves (3)
3  Set maxsectioncs = 0, S̃ = ∅ ;
4  for i = 1, 2, ..., M/log M do
5      for each subset C = {j_1, ..., j_|C|} of indices of columns in the ith section do
6          for each choice of (not necessarily distinct) characters, c_1, ..., c_|C| ∈ Σ do
7              Build a string s* of size M that has character c_k in position j_k (k = 1, ..., |C|) and in all other positions has a gap;
8              for each string s_t ∈ S do
9                  Find the alignment s*, s̃_t for the pair s*, s_t of maximum column score
10             end
11             if cs({s*, s̃_t}) = |C| for each t = 1, ..., m then
12                 if |C| > maxsectioncs then
13                     Set maxsectioncs = |C|, S̃ = {s̃_1, ..., s̃_m}
14                 end
15             end
16         end
17     end
18 end
19 return S̃
```

Correctness. For the correctness of the algorithm we show that an alignment achieving (3) is in the space of the solutions exhaustively explored by Section-Align.

Fix an alignment $\tilde{S} = \{\tilde{s}_1, \ldots, \tilde{s}_m\}$ satisfying (3). Let i^* be the section where $\tilde{S}$ achieves the maximum, and $C = \{j_1, \ldots, j_{|C|}\}$ be the set of columns in the ith section that are made of a single character and c_k $(k = 1, \ldots, |C|)$ be the only character in column j_k. Let s^* be the string of size M that has c_k in column j_k $(k = 1, \ldots, |C|)$ and gaps elsewhere. Then, s^* and $\tilde{s}_t$ match in all the positions in C, i.e., there is an alignment of s^* and s_t with column score equal to $|C|$. Therefore, $\tilde{S}$ is one of the possible solutions considered by Section-Align.

Time complexity. To show that the algorithm is polynomial time we first observe that each **for** loop is executed a number of times polynomial in M: the outermost one (line 4) is executed once for each of the $\frac{M}{\log M}$ sections, which we can upperbound as $O(M)$; for the second loop (line 5), we have that the number of possible sets of columns in a section is $O(2^{\log M}) = O(M)$; for the third loop (line 6), we observe that the number of choices of $|C|$ characters to associate to the columns of C is $O(\Sigma^{\log M}) = O(M^{\log |\Sigma|})$, which is also polynomial in M under the assumption of a constant size alphabet. The innermost **for** loop (line 8) is executed m times, one for each string in S. Each iteration requires to compute an alignment of s^* with a string s_i of maximum column score. This can be achieved in $O(M^2)$ time by a simple modification of the dynamic programming approach for the edit distance with the additional constraint that no additional gap can be inserted in s^*. Hence the cost of the innermost loop is $O(mM^2)$. This time bound also accounts for the cost of the last **if** (line 11). The final observation is that we can assume $M \leq \sum_{t=1}^{m} |s_i|$ since no alignment would need more columns than the number sufficient to completely separate the characters of each pair of strings. Therefore, the overall time complexity of the algorithm is $O\left(m\left(\sum_{i=1}^{m} |s_i|\right)^{4+\log|\Sigma|}\right)$ which is polynomial in the input size $\sum_{i=1}^{m} |s_i|$ (under the assumption of a constant alphabet size $|\Sigma|$).

Approximation guarantee. The approximation guarantee of this algorithm can be inferred as follows. Let $\hat{S}$ be the alignment with maximum column score for S, and let $OPT(S) = cs(\hat{S})$. Let $\hat{i} = \text{argmax}_i cs(\hat{S}, i)$ be the index of the section with the largest number of columns made of a single character in the optimal alignment $\hat{S}$. Hence, from (2)

$$cs(\hat{S}) \leq \frac{M}{\log M} cs(\hat{S}, \hat{i}). \tag{4}$$

Let $\tilde{S}$ be the alignment returned by Section-Align. Let $\tilde{S}_i$ be the alignment that among those built by Section-Align while processing the ith section (i.e., during the ith iteration of the outermost **for** loop) achieves the maximum column score in section i. Finally, let $\tilde{i} = \text{argmax}_i cs(\tilde{S}, i)$ be the index of the section where $\tilde{S}$ achieves the maximum score. By (3), we have

$$cs(\tilde{S}, \tilde{i}) \geq cs(\hat{S}, \hat{i}). \tag{5}$$

Since, for each i it holds that $cs(\tilde{S}) \geq cs(\tilde{S}, i)$, putting together (4) and (5), we have

$$cs(\tilde{S}) \geq cs(\tilde{S}, \tilde{i}) \geq cs(\hat{S}, \hat{i}) \geq \frac{cs(\hat{S})}{M/\log M} = \frac{OPT(S)}{M/\log M}.$$

From the above sequence of inequalities we have

$$\frac{OPT(S)}{cs(\tilde{S})} \leq \frac{M}{\log M},$$

thereby proving that SectionAlign is a $\frac{M}{\log M}$-approximation algorithm for BCS-Align.

4. Final remarks

We proved that computing the best multiple sequence alignment with respect to the column score is NP-hard and poly-APX-hard, even in a very restricted model where: strings are binary, we only try to optimize the number of columns of the alignment made of only zeroes, and we bound the number of gaps per row to 1. On the positive, algorithmic side, for the case where the alphabet of the input strings is bounded, we showed an $\frac{M}{\log M}$-approximation algorithm, almost matching the above inapproximability result.

There are some natural questions left open by our results and suggesting possible directions for future investigation. A first question regards the possibility of achieving better approximation guarantees for particular classes of instances (e.g., based on particular assumptions on the distribution of characters in the strings) in particular in the case of binary or quaternary (DNA) alphabets. Another interesting question is about the relationship between the column score and the SP score. In this direction, one might try to extend the model assuming that a column contributes to the column score not only when it is made of a single character but also if all characters in the column apart from a fixed number (or a fraction) of exceptions are equal. Both the case where the contribution of such an "almost mono-character" column is binary or proportional to the level of uniformity would be of interest, as they would constitute a sort of intermediate alignment criterium between the column score and the SPscore.

Finally, it would also be worth to investigate the relationship between the column score based multiple sequence alignment and parameterized variants of the Longest Common Supersequence (LCS) problem [10,11,13,14]—as noted in the introduction, LCS is a natural model of column score based multiple sequence alignment in the absence of any bounds on the number of gaps or restrictions on the uniform columns.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] S.J. Berkemer, C. Höner zu Siederdissen, P.F. Stadler, Compositional properties of alignments, Math. Comput. Sci. (2020).
[2] C. Notredame, Recent progress in multiple sequence alignment: a survey, Pharmacogenomics 3 (1) (2002) 131–144.
[3] J.D. Thompson, F. Plewniak, O. Poch, A comprehensive comparison of multiple sequence alignment programs, Nucleic Acids Res. 27 (13) (1999) 2682–2690.
[4] M. Zhang, W. Fang, J. Zhang, Z. Chi, Msaid: multiple sequence alignment based on a measure of information discrepancy, Comput. Biol. Chem. 29 (2) (2005) 175–181.
[5] H. Carrillo, D. Lipman, The multiple sequence alignment problem in biology, SIAM J. Appl. Math. 48 (5) (1988) 1073–1082.

[6] S.B. Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Biol. 48 (3) (1970) 443–453.
[7] L. Wang, T. Jiang, On the complexity of multiple sequence alignment, J. Comput. Biol. 1 (4) (1994) 337–348.
[8] M. Li, B. Ma, L. Wang, Finding similar regions in many sequences, J. Comput. Syst. Sci. 65 (1) (2002) 73–96.
[9] D. Maier, The complexity of some problems on subsequences and supersequences, J. ACM 25 (2) (1978) 322–336.
[10] C.S. Iliopoulos, M. Kubica, M.S. Rahman, T. Waleń, Algorithms for computing the longest parameterized common subsequence, in: B. Ma, K. Zhang (Eds.), Combinatorial Pattern Matching, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 265–273.
[11] C.S. Iliopoulos, M. Sohel Rahman, Algorithms for computing variants of the longest common subsequence problem, Theor. Comput. Sci. 395 (2) (2008) 255–267.
[12] D. Zuckerman, Linear degree extractors and the inapproximability of max clique and chromatic number, in: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, STOC '06, Association for Computing Machinery, New York, NY, USA, 2006, pp. 681–690.
[13] G. Blin, L. Bulteau, M. Jiang, P.J. Tejada, S. Vialette, Hardness of longest common subsequence for sequences with bounded run-lengths, in: J. Kärkkäinen, J. Stoye (Eds.), Combinatorial Pattern Matching, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 138–148.
[14] O. Keller, T. Kopelowitz, M. Lewenstein, On the longest common parameterized subsequence, in: Combinatorial Pattern Matching, Theor. Comput. Sci. 410 (51) (2009) 5347–5353.