Parameterized Complexity of the Arc-Preserving Subsequence Problem[⋆]

Dániel Marx[1] and Ildikó Schlotter[2]

[1] Tel Aviv University, Israel
[2] Budapest University of Technology and Economics, Hungary
{dmarx,ildi}@cs.bme.hu

Abstract. We study the Arc-Preserving Subsequence (APS) problem with unlimited annotations. Given two arc-annotated sequences P and T, this problem asks if it is possible to delete characters from T to obtain P. Since even the unary version of APS is NP-hard, we used the framework of parameterized complexity, focusing on a parameterization of this problem where the parameter is the number of deletions we can make. We present a linear-time FPT algorithm for a generalization of APS, applying techniques originally designed to give an FPT algorithm for Induced Subgraph Isomorphism on interval graphs [12].

1 Introduction

Many important problems in computational biology are related to pattern matching in strings, since DNA, RNA, or protein molecules can be viewed as sequences of nucleotides or amino acids. To gain information about such molecules, we often need to compare two sequences and measure their similarity.

Given two sequences S_1 and S_2 over some alphabet, the task of the Longest Common Subsequence (LCS) problem is to find the longest possible sequence that is the subsequence of both S_1 and S_2. In other words, we are looking for a sequence C that can be obtained both from S_1 and from S_2 by deleting characters. This problem arises in many applications, like deciding if two species are biologically related, or whether two proteins are likely to exhibit similar functionalities related to three-dimensional structure (protein folding). Another classical problem, Subsequence, asks if a sequence is the subsequence of another.

If we only want to deal with character sequences, LCS can be solved efficiently using dynamic programming. However, recent biological research suggests that we might loose relevant information if we model DNA, RNA, or protein molecules simply as sequences. The reason for this is that in such molecules, the shape and hence the functionality is greatly affected by chemical bonds between elements that might be far apart from each other in the sequence. *Arc-annotated sequences* are widely used to represent such bonds. In this model, any two elements (or *bases*) of a sequence can be connected to each other through an *arc*.

[⋆] Supported by ERC Advanced Grant DMMCA and by the Hungarian National Research Fund OTKA 67651.

For two arc-annotated sequences S_1 and S_2, the LONGEST ARC-PRESERVING COMMON SUBSEQUENCE or LAPCS asks for an arc-annotated sequence C of maximum length that can be obtained both from S_1 and from S_2 by deleting bases together with all arcs incident to them. Since LAPCS is NP-complete even if the arc structures are highly restricted [5, 6, 10], researchers focused on polynomial-time solvable cases and approximation algorithms [5, 10, 9, 11].

Another direction of research is to use the parameterized complexity framework [4, 7]. This area deals with NP-hard problems by giving algorithms that have an acceptable running time on many relevant instances. An algorithm is *fixed-parameter tractable* (FPT) if its running time is bounded by $f(k)n^{O(1)}$ for some function f, where n is the input size and k is the *parameter* associated with the input. The idea behind this definition is that the running time of an FPT algorithm remains tractable provided that the parameter has small value.

Parameterized complexity of LAPCS has already been studied, and FPT algorithms were presented for various parameterizations [1, 6]. An interesting parameterization is where the parameter is the number of deletions we are allowed to make in order to construct the common subsequence. This models a situation where we compare two sequences which are similar. An FPT algorithm was given in [1] with this parameter, but it only applies for a restricted case.

Unlike most previous results, we considered unlimited annotations where any two bases of a sequence can be connected by arcs. Instead of concentrating on LAPCS, we dealt with the more simple ARC-PRESERVING SUBSEQUENCE problem (APS), the annotated analog of SUBSEQUENCE. Given two arc-annotated sequences P and T, the task of APS is to find out whether the pattern sequence P can be obtained by deleting some bases of the target sequence T, together with all the arcs incident to them. We remark that APS on its own is an interesting problem in computation biology, and has been widely studied in the literature. Its NP-hardness has been proved for numerous restricted cases [2], and polynomial-time algorithms have been presented [8, 3] for limited arc structures.

Here, we present an FPT algorithm for the unlimited APS, where the parameter is the number k of deletions allowed. Our algorithm runs in $f(k)n$ time for some function f depending only on k, where n is the input size. In fact, we solve a generalization of APS where a few arcs can be deleted additionally. We mention that APS is W[1]-hard if the parameter is the length of the pattern [5].

The ideas and techniques applied here originate from an FPT algorithm solving a seemingly unrelated problem on interval graphs [12]. This algorithm answers the INDUCED SUBGRAPH ISOMORPHISM in FPT time: given two interval graphs G and H and a parameter k, is it possible to delete k vertices from G to obtains a graph isomorphic to H? Our work shows that research connected to interval graphs can be useful for arc-annotated sequences as well.

2 Problem definition and notation

We denote $\{1, \ldots, n\}$ by $[n]$. We refer to the elements of a sequence S over an alphabet Σ as *bases*. The i-th base of S is $S[i]$, and the length of S is $|S|$.

Let S_P and S_T be two sequences over Σ. Let $|S_P| = n_P$ and $|S_T| = n_T$, assume $n_P \leq n_T$. We say that S_P is a *subsequence* of S_T if S_P can be obtained by deleting bases from S_T, or equivalently, if there is a bijective mapping φ from $[n_P]$ into a subset of $[n_T]$ such that $\varphi(i_1) < \varphi(i_2)$ for each $1 \leq i_1 < i_2 \leq n_P$, and $S_P[i] = S_T[\varphi(i)]$ for each $i \in [n_P]$. We call such a φ an *alignment* of $(S_P; S_T)$. We write $S^{\text{del}}(\varphi)$ to denote the set of bases that have to be deleted from S_T according to φ, i.e. $S^{\text{del}}(\varphi) = [n_T] \setminus \bigcup_{i \in [n_P]} \varphi(i)$.

An *arc-annotation* A of a sequence S of length n is a multiset of pairs of integers from $[n]$, where each pair $(i_1, i_2) \in A$ satisfies $i_1 < i_2$. An *arc-annotated sequence* (S, A) is a sequence S together with an arc-annotation A for S. We say that an *arc* (i_1, i_2) *starts* at i_1, *ends* at i_2, and *connects* the positions i_1 and i_2 *incident* to it. We write $A(i_1, i_2)$ for the multiplicity of the pair (i_1, i_2) in A, and we write $A^+(i)$ and $A^-(i)$ for the set of arcs starting or ending at i, respectively. Also, we let a^{start} and a^{end} to denote the starting and ending position of an arc a. We use $|(S, A)|$ to denote the *size* of (S, A) in binary encoding.

Given two arc-annotated sequences (S_P, A_P) and (S_T, A_T), we say that (S_P, A_P) is an *arc-preserving subsequence* of (S_T, A_T) if it can be obtained from (S_T, A_T) by deleting bases from it, i.e. there is an alignment φ of $(S_P; S_T)$ such that $A_P(i, j) = A_T(\varphi(i), \varphi(j))$ for any $1 \leq i < j \leq |S_P|$. Such an alignment is an *arc-preserving alignment* of $(S_P, A_P; S_T, A_T)$. Note that by deleting a base, we also mean the deletion of the arcs incident to it. Given two arc-annotated sequence P and T, the Arc-Preserving Subsequence problem (APS) asks whether P is an arc-preserving subsequence of T.

We will deal with the following generalization of APS, which we call Almost APS or AAPS: given two arc-annotated sequences (S_P, A_P) and (S_T, A_T) and some $k_a \in \mathbb{Z}$, we ask if we can delete some bases from S_T (together with their incident arcs) and at most k_a arcs *in addition* to obtain (S_P, A_P). Formally, we have to decide if there is a set A^{del} of at most k_a arcs in A_T such that (S_P, A_P) is an arc-preserving subsequence of $(S_T, A_T \setminus A^{\text{del}})$. We call φ a *k_a-alignment* for $(S_P, A_P; S_T, A_T)$ if φ is an arc-preserving alignment of $(S_P, A_P; S_T, A_T \setminus A^*)$ for some set A^* with $|A^*| \leq k_a$. Also, we let $A^{\text{del}}(\varphi)$ to denote such an A^*.

Given a sequence S, let S^{rev} denote the reverse of S. For a position i of S, we will use i^{rev} to denote the position $|S| - i + 1$ of S^{rev} corresponding to i. If A is an arc-annotation of S, then let A^{rev} denote the corresponding arc-annotation of S^{rev}, meaning $A^{\text{rev}}(i_1, i_2) = A(i_2^{\text{rev}}, i_1^{\text{rev}})$. We also let $X^{\text{rev}} = \{i^{\text{rev}} \mid i \in X\}$ for any set X of positions in S.

If φ is a k_a-alignment for $(S_P, A_P; S_T, A_T)$, then φ^{rev} is the corresponding k_a-alignment for $(S_P^{\text{rev}}, A_P^{\text{rev}}; S_T^{\text{rev}}, A_T^{\text{rev}})$, i.e. $\varphi^{\text{rev}}(i) = (\varphi(i^{\text{rev}}))^{\text{rev}}$ for each i.

Due to lack of space, we omit several proofs, see the full paper for them.

3 Fixed-parameter tractability of APS

In this section we present an FPT algorithm for AAPS, a generalization of APS, with the parameterization where the parameters are the number of bases to delete and the number of arcs that can be deleted additionally.

Almost Arc-Preserving Subsequence
Input: Two arc-annotated sequences (S_P, A_P) and (S_T, A_T), and $k_a \in \mathbb{Z}$.
Parameters: k_a and $k_b = |S_T| - |S_P|$.
Task: decide whether (S_P, A_P) can be obtained from (S_T, A_T) by deleting k_b bases (together with their incident arcs) and k_a arcs in addition, i.e. whether there is a k_a-alignment φ for $(S_P, A_P; S_T, A_T)$.

Our aim is to prove the main result of the paper stated by Theorem 1.

Theorem 1. *There is an algorithm that solves any instance* $(S_P, A_P; S_T, A_T; k_a)$ *of the* Almost Arc-Preserving Subsequence *problem and runs in time* $k_b^{O(k_b^3 + k_b k_a)} |(S_T, A_T)|$ *where* $k_b = |S_T| - |S_P|$.

3.1 Outline of the algorithm

To prove Theorem 1, we present an algorithm that uses a bounded search tree technique in order to construct a k_a-alignment step by step. In certain situations, the algorithm might branch on a bounded number of possibilities to proceed with. Since both the number of such branchings and the possible directions of a branching will be bounded in terms of k_a and k_b, the size of the resulting search tree will be bounded by a function of k_a and k_b.

Actually, the algorithm described here has the following behavior: given an instance of APS, consisting of the arc-annotated sequences (S_P, A_P) and (S_T, A_T), and an integer k_a, it tries to construct a k_a-alignment φ for $(S_P, A_P; S_T, A_T)$. To do so, it fixes such a hypothetical solution φ, and looks for bases in $S^{\mathrm{del}}(\varphi)$ and arcs in $A^{\mathrm{del}}(\varphi)$, which we will call *removable bases* and *removable arcs* of φ, resp. More precisely, our algorithm does one of the followings in linear time:

– it produces an **arc-preserving alignment** ψ for $(S_P, A_P; S_T, A_T)$ (note that ψ is a k_a-alignment for $(S_P, A_P; S_T, A_T)$ as well),
– it correctly **rejects** the instance, or
– it produces a **removable base** or a **removable arc** of φ.

In the last case, we can delete the given base or arc, and apply the algorithm to the obtained instance. Notice that one of the parameters k_a and $k_b = |S_T| - |S_P|$ is decreased in the new instance. The presented algorithm will be shown to run in $f(k_a, k_b)|(S_T, A_T)|$ time for some functions f, which therefore implies Theorem 1 by proving that AAPS can be solved in $(k_a + k_b) f(k_a, k_b)|(S_T, A_T)|$ time.

Our algorithm might branch several times before producing an output as described above. Each such branch will be caused by guessing the answer to a question of the following form: given some position p in S_P, what is the value of the position $\varphi(p)$?[3] We interpret these branchings in the usual framework of bounded search trees: a branching happens when we do not know the exact value of a certain variable (such as the value of $\varphi(p)$ in the above example),

[3] In a few cases we will also need some additional branchings, described later on.

and thus we have to investigate every possible value. A certain branch examines one possible value of the variable, and it produces a correct output *if* the given variable indeed has the value associated with this branch. Since the examined cases always cover every possibilities, this implies that the output will be correct in at least one of the branches.

Although our algorithm seems to be a straightforward application of the bounded search tree methodology used frequently in parameterized algorithms, we had to overcome many difficulties to avoid any possibility of using an unbounded number of such guesses. The presented algorithm will apply considerably sophisticated methods to keep the search tree bounded.

3.2 Fragmentations and related concepts.

Fragmentation. To describe our knowledge of the partially constructed k_a-alignment we have, we introduce a data structure called *fragmentation.* By iteratively refining the fragmentation, we can get closer and closer to actually determine a k_a-alignment. We write $|S_P| = n_P$ and $|S_T| = n_T$.

Recall that φ is a fixed k_a-alignment for $(S_P, A_P; S_T, A_T)$. For some $1 \leq i_1 \leq i_2 \leq n_P$, we define the *block* $[i_1, i_2]$ in S_P to be the set of positions $i_1, i_1{+}1, \ldots, i_2$, and we define blocks in S_T similarly. Given a set of f disjoint blocks $\{[p_1^h, p_2^h] \mid h \in [f]\}$ in S_P and a set of f disjoint blocks $\{[t_1^h, t_2^h] \mid h \in [f]\}$ in S_T, we let $F_h = ([p_1^h, p_2^h], [t_1^h, t_2^h])$. We say that $\{F_h \mid h \in [f]\}$ is a *fragmentation* for φ, if

- $t_1^h \leq \varphi(p_1^h)$ and $\varphi(p_2^h) \leq t_2^h$ for each $h \in [f]$, and
- $p_1^{h+1} = p_2^h + 1$ and $t_1^{h+1} = t_2^h + 1$ for each $h \in [f-1]$.

We will call the element F_h for some $h \in [f]$ a *fragment.* We define $\sigma(F_h) = (t_2^h - t_1^h) - (p_2^h - p_1^h)$ and $\delta(F_h) = t_1^h - p_1^h$, which are both clearly non-negative integers. Note that $\delta(F_{h+1}) = \delta(F_h) + \sigma(F_h)$ holds for each $h \in [f-1]$. We say that a position $i \in [n_P]$ of S_P is *contained* in the fragment F_h, if $p_1^h \leq i \leq p_2^h$.

We will say that a fragment F is *trivial* if $\sigma(F)$ is zero, and *non-trivial* otherwise. We also call a position of S_P trivial (or non-trivial) in a fragmentation, if the fragment containing it is trivial (or non-trivial, resp). Given fragmentation for φ and a position i in S_P, we will use the notation $i_{\text{left}} = i + \delta(F)$ and $i_{\text{right}} = i + \delta(F) + \sigma(F)$, where F is the fragment containing i. Observe that

$$i_{\text{left}} \leq \varphi(i) \leq i_{\text{right}}$$

always holds. We will classify a position i of S_P as follows:

- If $\varphi(i) = i_{\text{left}}$, then i is *left-aligned.*
- If $\varphi(i) = i_{\text{right}}$, then i is *right-aligned.*
- If $\varphi(i) = j$ such that $i_{\text{left}} < j < i_{\text{right}}$, then i is *skew.*

If i is trivial, then only $\varphi(i) = i_{\text{left}} = i_{\text{right}}$ is possible. Thus, each trivial position must be both left- and right-aligned.

Notice that each fragment F must contain exactly $\sigma(F)$ positions that are contained in $S^{\text{del}}(\varphi)$. This implies the following bounds.

Proposition 2. *If $\mathcal{F}$ is a fragmentation for φ, then $\sum_{F\in\mathcal{F}} \sigma(F) = k_b$. In particular, $\mathcal{F}$ can have at most k_b non-trivial fragments.*

A *marked fragmentation* for φ is a pair $(\mathcal{F}, M)$ formed by a fragmentation $\mathcal{F}$ for φ and a set M of positions in S_P such that each $m \in M$ is a trivial position in $\mathcal{F}$. We say that the trivial positions contained in M are *marked*.

For a fragment $F = ([p_1, p_2], [t_1, t_2])$ we let $F^{\text{rev}} = ([p_2^{\text{rev}}, p_1^{\text{rev}}], [t_2^{\text{rev}}, t_1^{\text{rev}}])$, hence a fragmentation $\mathcal{F}$ for φ clearly yields a fragmentation $\mathcal{F}^{\text{rev}} = \{F^{\text{rev}} | F \in \mathcal{F}\}$ for φ^{rev} as well. Note that if a position i of S_P is left-aligned (right-aligned) in $\mathcal{F}$, then the position i^{rev} is right-aligned (left-aligned, resp.) in $\mathcal{F}^{\text{rev}}$.

Pairing arcs. Given a position i in S_P, let us order the arcs c in $A_P^+(i)$ increasingly according to their right endpoint c^{end}. Similarly, we order the arcs in $A_P^-(i)$ increasingly according their left endpoint. In both cases, we break ties arbitrarily. Also, we order the arcs in $A_T^+(j)$ and $A_T^-(j)$ in the same way for each position j in S_T. Now, we "pair" arcs in $A_P^+(i)$ with arcs in $A_T^+(i_{\text{left}})$, and also arcs in $A_P^-(i)$ with arcs in $A_T^-(i_{\text{left}})$ according to their ranking in this ordering. To this end, we construct the sets $R_{\text{left}}^+(i) \subseteq A_P^+(i) \times A_T^+(i_{\text{left}})$ and $R_{\text{left}}^-(i) \subseteq A_P^-(i) \times A_T^-(i_{\text{left}})$ in the following way. We put a pair (c, d) into $R_{\text{left}}^+(i)$, if $c \in A_P^+(i)$, $d \in A_T^+(i_{\text{left}})$, and c has the same rank (according to the above ordering) in $A_P^+(i)$ as the rank of d in $A_T^+(i_{\text{left}})$. Similarly, we put a pair (c, d) into $R_{\text{left}}^-(i)$, if $c \in A_P^-(i)$, $d \in A_T^-(i_{\text{left}})$, and c has the same rank in $A_P^-(i)$ as the rank of d in $A_T^-(i_{\text{left}})$. In addition, we define the sets $R_{\text{right}}^+(i)$ and $R_{\text{right}}^-(i)$ analogously, by substituting i_{right} for i_{left} in the above definitions. The key properties of these sets are summarized below.

Lemma 3. *We know $\varphi(c^{\text{end}}) = d^{\text{end}}$ and $\varphi(c^{\text{start}}) = d^{\text{start}}$ in the following cases:*
(1) If $(c, d) \in R_{\text{left}}^+(i)$ and $|A_P^+(i)| = |A_T^+(i_{\text{left}})|$ for some left-aligned i.
(2) If $(c, d) \in R_{\text{left}}^-(i)$ and $|A_P^-(i)| = |A_T^-(i_{\text{left}})|$ for some left-aligned i.
(3) If $(c, d) \in R_{\text{right}}^+(i)$ and $|A_P^+(i)| = |A_T^+(i_{\text{right}})|$ for some right-aligned i.
(4) If $(c, d) \in R_{\text{right}}^-(i)$ and $|A_P^-(i)| = |A_T^-(i_{\text{right}})|$ for some right-aligned i.

Arcs connecting two non-trivial fragments. Given two non-trivial fragments F and H of a fragmentation with F preceding H, we define three disjoint subsets of those arcs of A_P that start in a position of F and end in a position of H. These sets will be denoted by $\mathcal{L}(F, H)$, $\mathcal{R}(F, H)$, and $\mathcal{X}(F, H)$, and we construct them as follows. Suppose that $c = (f, h) \in A_P$ for some f and h contained in F and H, respectively. We put c in exactly one of these three sets, if $(c, d) \in R_{\text{left}}^-(h)$ for some arc $d \in A_T$ such that $f_{\text{left}} \leq d^{\text{start}} \leq f_{\text{right}}$. If $d^{\text{start}} = f_{\text{left}}$ then we put c into $\mathcal{L}(F, H)$, if $d^{\text{start}} = f_{\text{right}}$ then we put c into $\mathcal{R}(F, H)$, and if $f_{\text{left}} < d^{\text{start}} < f_{\text{right}}$ then we put c into $\mathcal{X}(F, H)$.

By Lemma 3, if the positions in H are left-aligned, then the left endpoints of the arcs in $\mathcal{R}(F, H)$ must be right-aligned. Similarly, the left endpoints of the arcs in $\mathcal{X}(F, H)$ must be skew in such a case. Proposition 4 states these observations in a precise manner. Since we would like to ensure each position to be left-aligned, we will try to get rid of the arcs in $\mathcal{R}(F, H)$ and $\mathcal{X}(F, H)$.

Proposition 4. *Let i be left-aligned, $|A^-_P(i)| = |A^-_T(i_{\text{left}})|$, and $c \in A^-_P(i)$.*
(1) If $c \in \mathcal{L}(F,H)$, then c^{start} is left-aligned.
(2) If $c \in \mathcal{R}(F,H)$, then c^{start} is right-aligned.
(3) If $c \in \mathcal{X}(F,H)$, then c^{start} is skew.

We say that two positions $f_1, f_2 \in [n_P]$ are *conflicting* for (F,H), if $f_1 \leq f_2$, $A^+_P(f_1) \cap \mathcal{R}(F,H) \neq \emptyset$ and $A^+_P(f_2) \cap \mathcal{L}(F,H) \neq \emptyset$. In such a case, we say that any $h \geq \max\{h_1, h_2\}$ in H is *conflict-inducing* for (F,H) (and for the conflicting pair (f_1, f_2)), where h_1 denotes the minimal position for which $(f_1, h_1) \in \mathcal{R}(F,H)$, and h_2 denotes the minimal position for which $(f_2, h_2) \in \mathcal{L}(F,H)$. Notice that if such a conflict-inducing h is left-aligned, then both h_1 and h_2 are left-aligned. By Proposition 4, this implies that f_1 is right-aligned and f_2 is left-aligned. But since f_1 precedes f_2, this cannot happen. This implies the following observation.

Proposition 5. *If a position h is conflict-inducing for (F,H) in a given fragmentation, then h cannot be left-aligned.*

In addition, if $\mathcal{L}(F,H) \neq \emptyset$, then let $L^{\max}(F,H)$ denote the largest position f in F for which $A^+_P(f) \cap \mathcal{L}(F,H) \neq \emptyset$. Let the *L-critical position for* (F,H) be the smallest position h contained in H for which $(L^{\max}(F,H), h) \in \mathcal{L}(F,H)$. Similarly, if $\mathcal{R}(F,H) \neq \emptyset$, then let $R^{\min}(F,H)$ denote the smallest position f in F for which $A^+_P(f) \cap \mathcal{R}(F,H) \neq \emptyset$. Also, let the *R-critical position for* (F,H) be the smallest position h in H for which $(R^{\min}(F,H), h) \in \mathcal{R}(F,H)$.

Now, a position h in H is *LR-critical for* (F,H), if either h is the R-critical position for (F,H) and $\mathcal{L}(F,H) = \emptyset$, or $h = \max\{h_L, h_R\}$ where h_L is the L-critical and h_R is the R-critical position for (F,H). Note that both cases require $\mathcal{R}(F,H) \neq \emptyset$. Moreover, H contains an LR-critical position for (F,H), if and only if $\mathcal{R}(F,H) \neq \emptyset$. Intuitively, if an LR-critical position in H is left-aligned, then this implies that some position in F is right-aligned.

Note that the definitions of the sets $\mathcal{L}(F,H), \mathcal{R}(F,H)$, and $\mathcal{X}(F,H)$ together with the definitions connected to them as described above depend on the given fragmentation, so whenever the fragmentation changes, these must be adjusted appropriately as well. (In particular, arcs in $\mathcal{L}(F,H), \mathcal{R}(F,H)$, and $\mathcal{X}(F,H)$ must start and end in two different non-trivial fragments.)

Properties 1-9. Let $(\mathcal{F}, M)$ be a marked fragmentation for φ. Our aim is to ensure that the properties given below hold for each position in S_P. Intuitively, these properties mirror the expectation that every position should be left-aligned. Note that although we cannot decide whether $(\mathcal{F}, M)$ is a correct marked fragmentation without knowing the k_a-alignment φ, we are able to check whether these properties hold for some position i in $(\mathcal{F}, M)$.

Property 1: $S_P[i] = S_T[i_{\text{left}}]$.
Property 2: If i is non-trivial, then $|A^+_P(i)| = |A^+_T(i_{\text{left}})|$ and $|A^-_P(i)| = |A^-_T(i_{\text{left}})|$.
Property 3: If i is non-trivial, then $A_P(y,i) = A_T(y_{\text{left}}, i_{\text{left}})$ for any $y < i$ contained in the same fragment as i.
Property 4: If i is non-trivial, then for every $(c,d) \in R^+_{\text{left}}(i)$ such that $c^{\text{end}} = y$ is non-trivial, $y_{\text{left}} \leq d^{\text{end}} \leq y_{\text{right}}$ holds. Also, for every $(c,d) \in R^-_{\text{left}}(i)$ such that $c^{\text{start}} = y$ is non-trivial, $y_{\text{left}} \leq d^{\text{start}} \leq y_{\text{right}}$ holds.

Property 5: No arc in $\mathcal{X}(F,H)$ for some (F,H) ends at i.
Property 6: i is not conflict-inducing for any (F,H).
Property 7: i is not LR-critical for any (F,H).
Property 8: If i is non-trivial, then for every $(c,d) \in R^{+}_{\text{left}}(i)$ such that $c^{\text{end}} = y$ is non-trivial, $d^{\text{end}} = y_{\text{left}}$ holds. Also, for every $(c,d) \in R^{-}_{\text{left}}(i)$ such that $c^{\text{start}} = y$ is non-trivial, $d^{\text{start}} = y_{\text{left}}$ holds.
Property 9: If i is non-trivial, then for each marked position $m \in M$, $A_P(i,m) = A_T(i_{\text{left}}, m_{\text{left}})$ holds if $m > i$, and $A_P(m,i) = A_T(m_{\text{left}}, i_{\text{left}})$ holds if $m < i$.

Observe that each of these properties depend on the fragmentation $\mathcal{F}$, and Property 9 depends on the set of marked positions M as well. Also, if some property holds for a position i in $(\mathcal{F}, M)$, then this does not imply that the property holds for i^{rev} in $(\mathcal{F}^{\text{rev}}, M^{\text{rev}})$, as most of these properties are not symmetric. For example, i_{left} and i_{right} both have a different meaning in the fragmentation $\mathcal{F}$ and in $\mathcal{F}^{\text{rev}}$. We say that a position $i \in [n_P]$ *violates* Property ℓ $(1 \leq \ell \leq 9)$ in a marked fragmentation $(\mathcal{F}, M)$, if Property ℓ does not hold for i in $(\mathcal{F}, M)$.

If the first eight properties hold for each position both in $(\mathcal{F}, M)$ and in $(\mathcal{F}^{\text{rev}}, M^{\text{rev}})$, then we say that $(\mathcal{F}, M)$ is *8-proper.* We say that $(\mathcal{F}, M)$ is *proper*, if it is 8-proper and Property 9 holds hold for each position of S_P in $(\mathcal{F}, M)$. Note that we do not care whether Property 9 holds for the positions in the reversed instance, so $(\mathcal{F}, M)$ is proper even if Property 9 does not hold in $(\mathcal{F}^{\text{rev}}, M^{\text{rev}})$.

3.3 Description of the algorithm

We start with a marked fragmentation where $M = \emptyset$ and the fragmentation contains only the unique fragment $([1, n_P], [1, n_T])$, which is non-trivial if $k_b > 0$. Given a marked fragmentation $(\mathcal{F}, M)$, we do the following: if one of Properties $1, 2, \ldots, 9$ does not hold for some position i in $(\mathcal{F}, M)$ or one of the first eight properties does not hold for some i in the reversed marked fragmentation $(\mathcal{F}^{\text{rev}}, M^{\text{rev}})$, then we will either **reject** the instance, output a **removable base** of φ, or modify the given marked fragmentation. If the given marked fragmentation is proper, the algorithm returns an output using Lemmas 9 and 10.

To do this, in each step we choose the first property violated by a position either in $(\mathcal{F}, M)$ or in $(\mathcal{F}^{\text{rev}}, M^{\text{rev}})$. Observe that we can assume w.l.o.g. that there is an ℓ $(1 \leq \ell \leq 9)$ such that Properties $1, \ldots, \ell - 1$ hold for each position both in $(\mathcal{F}, M)$ and in $(\mathcal{F}^{\text{rev}}, M^{\text{rev}})$, but Property ℓ is violated by a position in S_P in $(\mathcal{F}, M)$, otherwise we simply reverse the instance. (We only reverse it if this condition is not true.)

Given ℓ, the algorithm takes the first position i violating Property ℓ, and branches on choosing $\varphi(i)$ according to $i_{\text{left}} \leq \varphi(i) \leq i_{\text{right}}$. By Proposition 2, this results in at most $k_b + 1$ directions. Next, the algorithm handles each of the cases in a different manner, according to whether i turns out to be left-aligned, right-aligned, or skew. We consider these cases in a general way that is essentially independent from ℓ, and mainly relies on the type of i. We suppose that i is contained in a fragment $F^i = ([p_1, p_2], [t_1, t_2])$.

Extremal cases. Assume that $i = p_1$ and i is skew or right-aligned, or $i = p_2$ and i is skew or left-aligned. In these cases, we can find at least one **removable base** of φ. First, if $i = p_1$ and i is skew or right-aligned, then each base $S_T[j]$ must be deleted for each j where $t_1 \leq j < \varphi(i)$. Second, if $i = p_2$ and i is skew or left-aligned, then $S_T[j]$ must be deleted for each j where $\varphi(i) < j \leq t_2$.

Skew position. Suppose that $i > p_1$ and j is skew, meaning that $\varphi(i) = j$ for some j with $i_{\text{left}} < j < i_{\text{right}}$. In this case, we can divide the fragment F^i, or more precisely, we can delete F^i from the fragmentation $\mathcal{F}$ and add the new fragments $([p_1, i-1], [t_1, j-1])$ and $([i, p_2], [j, t_2])$. Note that the newly introduced fragments are non-trivial by the bounds on j. We also modify M by declaring every trivial position of the fragmentation to be marked (no matter whether it was marked or not before). Observe that the number of non-trivial fragments increases in this step. By Proposition 2, this can happen at most $k_b - 1$ times.

Left-aligned position. Lemma 6 summarizes our results that show how to deal with the case when i is left-aligned and $i < p_2$. The proof of this lemma is essential in the correctness of our algorithm.

Lemma 6. *Suppose that Property ℓ ($1 \leq \ell \leq 9$) does not hold for some $i \in [n_P]$ in the marked fragmentation $(\mathcal{F}, M)$, but all the previous properties hold for each position both in $(\mathcal{F}, M)$ and in $(\mathcal{F}^{\text{rev}}, M^{\text{rev}})$. If i is left-aligned, then depending on ℓ, we can do one of the followings in linear time (without any branchings):*

A) reject correctly,
B) output a removable arc of φ,
C) find that i is incident to a removable arc of φ (this only happens if $\ell = 2$),
D) produce a skew position i', or
E) produce a set N of at most $2k_b - 1$ positions in S_T such that $N \cap S^{\text{del}}(\varphi) \neq \emptyset$.

In Case A or B, we **reject** or output a **removable arc** of φ.

In Case C, we put the non-trivial position i in a set W, which will only store positions in S_T that are incident to a removable arc of φ. (We set $W = \emptyset$ initially.) Whenever Case C happens, we examine whether $|W| \leq 2k_a$. If not, then we **reject** the input. This is correct, since there can be at most k_a removable arcs of φ, and each such arc is incident to two bases.

If $|W| \leq 2k_a$ holds, then we modify the given fragmentation, replacing F^i by new fragments $F_1 = ([p_1, i], [t_1, i_{\text{left}}])$ and $F_2 = ([i+1, p_2], [i_{\text{left}} + 1, t_2])$. By $\varphi(i) = i_{\text{left}}$, this yields a fragmentation for φ. Note that F_1 is trivial and F_2 is non-trivial. We mark each position of F_2, putting them into M. We refer to this operation as a *left split* at i. Since i becomes trivial in F_1, each position can be placed into W at most once. Thus, Case C can happen at most $2k_a$ times without rejecting.

In Cases D and E, we might branch into a bounded number of additional branches. In Case D, we branch on those choices of $\varphi(i')$ where i' is indeed skew, which means $\sigma(F^i) - 1 \leq k_b - 1$ directions, and we handle each branch according to the way described above (dividing one fragment at the skew position i'). In Case E, we branch into at most $2k_b - 1$ directions on choosing a **removable base** of φ from N and outputting it.

Note that Case D or E can happen at most k_b times, by our observation that a skew position can only be found at most $k_b - 1$ times.

We remark that if i is trivial, then we treat it as left-aligned.

Right-aligned position. Suppose that $i > p_1$ and i is right-aligned. In this case, we replace F^i by new fragments $F_1 = ([p_1, i-1], [t_1, i_{\text{right}} - 1])$ and $F_2 = ([i, p_2], [i_{\text{right}}, t_2])$. This yields a fragmentation where F_1 is non-trivial and F_2 is trivial. We refer to this operation as performing a *right split* at j. If this happens because i violated Property ℓ for some $\ell \leq 8$, then we mark every trivial position (including those contained in F_2), by putting them into M. If $\ell = 9$, then we do not modify M, so the trivial positions of F_2 will not be marked.

The above process either produces a **removable base** of φ, **rejects** correctly, or ends by providing a marked fragmentation that is proper. In the remaining steps of the algorithm, the set M will never be modified, and the only possible modification of the actual fragmentation will be to perform a right split.

Given a proper marked fragmentation $(\mathcal{F}, M)$, we make use of Lemma 9 below. This lemma gives sufficient conditions to do one of the followings.

- Find out that some non-trivial position i is right-aligned. In this case, we perform a right split at i in the actual fragmentation.
- Find a **removable arc** of φ.
- **Reject** correctly.

Our algorithm applies Lemma 9 repeatedly, until it either stops (by rejecting or outputting a removable arc of φ), or finds that none of the conditions of Lemma 9 apply. Before stating this lemma, we need two more important observations. First, Lemma 7 shows that the repeated application of Lemma 9 results in a proper fragmentation. Second, Lemma 8 states some useful invariants that hold for each fragmentation obtained by us after a proper fragmentation is achieved.

Lemma 7. *If $(\mathcal{F}, M)$ is proper and $\mathcal{F}'$ is obtained by applying an arbitrary number of right splits to $\mathcal{F}$, then $(\mathcal{F}', M)$ is proper as well.*

Lemma 8. *Let $(\mathcal{F}, M)$ be a 8-proper marked fragmentation whose trivial positions are all marked. Suppose that $\mathcal{F}'$ is obtained by applying an arbitrary number of right splits to the fragmentation $\mathcal{F}$.*
(1) For each i that is not marked ($i \in [n_P] \setminus M$), both $A_P^+(i) = A_T^+(i_{\text{right}})$ and $A_P^-(i) = A_T^-(i_{\text{right}})$ hold in $(\mathcal{F}', M)$.
(2) Suppose that neither i nor j is marked ($i, j \in [n_P] \setminus M$) and $c = (i, j) \in A_P$. If $(c, d) \in R^+_{\text{right}}(i)$ for some $d \in A_T^+(i_{\text{right}})$, then $d^{\text{end}} = j_{\text{right}}$. Similarly, if $(c, d) \in R^-_{\text{right}}(j)$ for some $d \in A_T^-(j_{\text{right}})$, then $d^{\text{start}} = i_{\text{right}}$.

Now, we can state Lemma 9.

Lemma 9. *Let $(\mathcal{F}, M)$ be a proper marked fragmentation for φ obtained by our algorithm, and let $a, b \in [n_P]$.*
(i) Suppose that a is trivial but not marked and b is non-trivial. If $(a, b) \in A_P$

or $(b,a) \in A_P$, then b is right-aligned.
(ii) If a and b are trivial, $a < b$ and $A_P(a,b) \neq A_T(a_{\text{left}}, b_{\text{left}})$, then we can either ***reject*** *or output a* ***removable arc*** *of φ.*

After applying Lemma 9 repeatedly, the algorithm either stops by rejecting or outputting a removable arc of φ, or it finds that neither of the conditions (i) and (ii) of Lemma 9 holds. Let $(\mathcal{F}, M)$ be the final marked fragmentation obtained. Note that the algorithm does not modify the set M of marked trivial positions when applying Lemma 9, and it can only modify the actual fragmentation by performing a right split. Hence, Lemma 7 yields that $(\mathcal{F}, M)$ is proper.

Using $(\mathcal{F}, M)$, Lemma 10 claims that we can find an arc-preserving alignment for $(S_P, A_P; S_T, A_T)$ in linear time. Hence, the final step of our algorithm, finishing its description, is to output this **arc-preserving alignment**.

Lemma 10. *Let $(\mathcal{F}, M)$ be a proper marked fragmentation for φ obtained by the algorithm. If none of the conditions of Lemma 9 holds, then we can produce an* ***arc-preserving alignment*** *ψ for $(S_P, A_P; S_T, A_T)$ in linear time.*

Proof. We show that defining $\psi(i) = i_{\text{left}}$ for each position $i \in [n_P]$ fulfills the requirements. For this, we have to prove $S_P[i] = S_T[i_{\text{left}}]$ for each position $i \in [n_P]$, and $A_P(i,j) = A_T(i_{\text{left}}, j_{\text{left}})$ for each two positions $i \neq j \in [n_P]$.

First, as Property 1 holds for each position in $\mathcal{F}$, we know $S_P[i] = S_T[i_{\text{left}}]$ for each $i \in [n_P]$. It remains to show $A_P(i,j) = A_T(i_{\text{left}}, j_{\text{left}})$ for each $i \neq j \in [n_P]$. If both i and j are trivial positions, then this is true because the conditions of (ii) in Lemma 9 do not apply. If both i and j are non-trivial, then $A_P(i,j) = A_T(i_{\text{left}}, j_{\text{left}})$ again holds, by Properties 2 and 8 for j. Now, if i is non-trivial but j is trivial and marked (or vice versa), then Property 9 implies the required equality. Finally, if one of i and j is non-trivial and the other one is trivial but not marked, then $A_P(i,j) = 0$ holds, since (i) of Lemma 9 is not applicable. □

3.4 Analysis of the algorithm

In this section, we give some hints how to analyse the running time of the presented algorithm. The following lemma, stating the key properties of the our algorithm, proves Theorem 1.

Lemma 11. *Let $(S_P, A_P, S_T, A_T, k_a)$ be the given instance of* APS*. The presented algorithm branches into at most $f(k_a, k_b)$ directions in total for some function f such that in each branch it does one of the followings (supposing that the conditions of the given branch do hold):*

- *it gives an* ***arc-preserving alignment*** *ψ of $(S_P, A_P; S_T, A_T)$,*
- *it correctly* ***rejects*** *the instance, or*
- *it outputs a* ***removable base*** *or a* ***removable arc*** *of φ.*

Moreover, each branch takes linear time in the size of the input.

Although we do not prove Lemma 11 due to lack of space, we give the most important definitions used in the proof.

Given a fragmentation $\mathcal{F}$ for φ, a fragment $F \in \mathcal{F}$, and some ℓ ($1 \leq \ell \leq 8$), let $\pi(\mathcal{F}, F, \ell)$ be 1 if Property ℓ holds for each position i in F, and 0 otherwise. Let $N(\mathcal{F})$ denote the set of non-trivial fragments in $\mathcal{F}$. We define the *measure* $\mu(\mathcal{F})$ of a given fragmentation $\mathcal{F}$ for φ as follows:

$$\mu(\mathcal{F}) = \sum_{1 \leq \ell \leq 8} \Bigg(\sum_{F \in N(\mathcal{F})} \pi(\mathcal{F}, F, \ell) + \sum_{F \in N(\mathcal{F}^{\mathrm{rev}})} \pi(\mathcal{F}^{\mathrm{rev}}, F, \ell) \Bigg).$$

Note that $\mu(\mathcal{F}) = \mu(\mathcal{F}^{\mathrm{rev}})$ is trivial, so reversing a fragmentation does not change its measure. The importance of this definition is shown by Lemma 12.

Lemma 12. *Let $\mathcal{F}_1, \ldots, \mathcal{F}_t, \mathcal{F}_{t+1}$ be a series a fragmentations such that for each $i \in [t]$ the algorithm obtains $\mathcal{F}_{i+1}$ from $\mathcal{F}_i$ by applying a left or a right split at a position j_i violating Property ℓ_i in $\mathcal{F}_i$. Then (1) $\mu(\mathcal{F}_{i+1}) \geq \mu(\mathcal{F}_i)$ for each $i \in [t]$, and (2) if $\mu(\mathcal{F}_1) = \mu(\mathcal{F}_t)$, then $t \leq k_b$ holds.*

References

1. J. Alber, J. Gramm, J. Guo, and R. Niedermeier. Computing the similarity of two sequences with nested arc annotations. *Theor. Comput. Sci.*, 312(2-3):337–358, 2004.
2. G. Blin, G. Fertin, R. Rizzi, and S. Vialette. What makes the Arc-Preserving Subsequence problem hard? In *IWBRA'05: Proceedings of the 5th Int. Workshop on Bioinformatics Research and Applications*, volume 3515 of *Lecture Notes in Computer Science*, pages 860–868. Springer-Verlag, 2005.
3. P. Damaschke. A remark on the subsequence problem for arc-annotated sequences with pairwise nested arcs. *Inf. Process. Lett.*, 100(2):64–68, 2006.
4. R. G. Downey and M. R. Fellows. *Parameterized complexity.* Monographs in Computer Science. Springer-Verlag, New York, 1999.
5. P. A. Evans. *Algorithms and complexity for annotated sequence analysis.* PhD thesis, University of Victoria, Canada, 1999.
6. P. A. Evans. Finding common subsequences with arcs and pseudoknots. In *CPM '99: Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching*, volume 1645 of *Lecture Notes in Computer Science*, pages 270–280. Springer-Verlag, 1999.
7. J. Flum and M. Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, New York, 2006.
8. J. Gramm, J. Guo, and R. Niedermeier. Pattern matching for arc-annotated sequences. *ACM Trans. Algorithms*, 2(1):44–65, 2006.
9. T. Jiang, G. Lin, B. Ma, and K. Zhang. The longest common subsequence problem for arc-annotated sequences. *J. Discrete Algorithms*, 2(2):257–270, 2004.
10. G. Lin, Z.-Z. Chen, T. Jiang, and J. Wen. The longest common subsequence problem for sequences with nested arc annotations. *J. Comput. Syst. Sci.*, 65(3):465–480, 2002.
11. B. Ma, L. Wang, and K. Zhang. Computing similarity between rna structures. *Theor. Comput. Sci.*, 276(1-2):111–132, 2002.
12. D. Marx and I. Schlotter. Cleaning interval graphs. *CoRR*, abs/1003.1260, 2010. arXiv:1003.1260 [cs.DS].