The Chain Alignment Problem

Leandro Lima[1,2][0000−0001−8976−2762] and Said Sadique Adi[3][0000−0002−0349−7441]

[1] Univ Lyon, Université Lyon 1, CNRS, Laboratoire de Biométrie et Biologie Evolutive UMR5558 F-69622 Villeurbanne, France.
[2] EPI ERABLE - Inria Grenoble, Rhône-Alpes, France.
[3] Universidade Federal de Mato Grosso do Sul, Faculdade de Computação, 79070-900, Campo Grande, Brazil.
leandro.ishi-soares-de-lima@inria.fr, said@facom.ufms.br

Abstract. This paper introduces two new combinatorial optimization problems involving strings, namely, the Chain Alignment Problem, and a multiple version of it, the Multiple Chain Alignment Problem. For the first problem, a polynomial-time algorithm using dynamic programming is presented, and for the second one, a proof of its $\mathcal{NP}$-hardness is provided and some heuristics are proposed for it. The applicability of both problems here introduced is attested by their good results when modeling the Gene Identification Problem.

Keywords: Chain Alignment Problem, dynamic programming, $\mathcal{NP}$-hardness, Gene prediction

1 Introduction

Problems involving strings can be found in many theoretical and practical areas, such as molecular biology, pattern search, text editing, data compression, etc. The present paper proposes two new combinatorial optimization problems involving strings: the Chain Alignment Problem and a multiple version of it, called the Multiple Chain Alignment Problem. Both can serve as models for other problems in several research areas. In Bioinformatics, for example, they can be used as a combinatorial optimization formulation for the gene prediction task [9] and for the problem of protein comparison by local structure segments [18]. Other applications include automated text categorization [16] and sequence data clustering [7]. The Chain Alignment Problem is closely related to a well-known problem in Bioinformatics, namely the Spliced Alignment Problem [4].

This paper explores the theoretical properties and algorithmic solutions for the Chain Alignment Problem and the Multiple Chain Alignment Problem. More specifically, for the first problem, a polynomial-time algorithm using dynamic programming is presented. For the second problem, it is proved that it is very unlikely that there exists a polynomial-time algorithm for it, i.e., we show it is $\mathcal{NP}$-hard through a reduction from the Longest Common Subsequence Problem. Given the $\mathcal{NP}$-hardness of the Multiple Chain Alignment Problem, we propose three different heuristics for it.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

Despite our focus being a theoretical study of the problems introduced here, we also show that they can be applied to practical problems. More specifically, we model a variant of the Gene Identification Problem as the Chain Alignment Problem and as the Multiple Chain Alignment Problem, and present some results which attest their applicability.

This paper is structured as follows. Section 2 formally defines the Chain Alignment Problem and the Multiple Chain Alignment Problem. Section 3 describes an efficient dynamic programming algorithm for the Chain Alignment Problem. Section 4 is devoted to the proof of the $\mathcal{NP}$-hardness of the Multiple Chain Alignment Problem and Section 5 to a brief description of three heuristics for this problem. Section 6 discusses the application of both problems in the Gene Identification Problem. Finally, some concluding remarks and future directions for research are discussed in Section 7.

2 The Chain Alignment Problem

For a better understanding of the Chain Alignment Problem, consider the following definitions. An *alphabet* is a finite set of *symbols*[1]. Let $s = s[1]s[2]\dots s[n]$ be a finite string over an alphabet Σ, whose *length* is denoted by $|s| = n$. The *empty string*, whose length equals to zero, is denoted by ϵ. We say that a string $s' = s'[1]s'[2]\dots s'[m]$ is a *subsequence* of s if there exists a strictly increasing sequence $\mathcal{I} = i_1, i_2, \dots, i_m$ of indices of s such that $s[i_j] = s'[j]$ for all $1 \leq j \leq m$. If $i_{k+1} - i_k = 1$ for all $1 \leq k < m$, we call s' a *segment* or a *substring* of s.

Further, let $b = s[i]\dots s[j]$ be a segment of a string s. The position of the first (last) symbol of b in s is denoted by $first(b) = i$ ($last(b) = j$). Let $\mathcal{B} = \{b_1, b_2, \dots, b_u\}$ be a set of u segments of s. $\mathcal{B}$ is defined as an *ordered set of segments* if: 1) $first(b_i) < first(b_{i+1})$ or 2) $first(b_i) = first(b_{i+1})$ and $last(b_i) < last(b_{i+1})$, for $1 \leq i \leq u-1$. Moreover, a segment $b' = s[i]\dots s[j]$ of s *overlaps* another segment $b'' = s[k]\dots s[l]$ of s if $k \leq i \leq l$, or $k \leq j \leq l$, or $i \leq k \leq j$, or $i \leq l \leq j$. On the other hand, if $j < k$, we say that b' *precedes* b'', and this relation is denoted by $b' \prec b''$. Using the previous definitions, a *chain* $\Gamma_{\mathcal{B}}$ of an ordered set of segments $\mathcal{B}$ is defined as a subset $\Gamma_{\mathcal{B}} = \{b_i, b_j, \dots, b_p\}$ of $\mathcal{B}$ such that $b_i \prec b_j \prec \dots \prec b_p$. In addition, the string resulting from the concatenation of the segments of a chain $\Gamma_{\mathcal{B}}$ is denoted by $\Gamma^{\bullet}_{\mathcal{B}}$, i.e., $\Gamma^{\bullet}_{\mathcal{B}} = b_i \bullet b_j \bullet \dots \bullet b_p$, where $\bullet$ is the string concatenation operator.

Finally, given two strings s and t, $sim_{\omega}(s,t)$ denotes the *similarity* (or the score of an *optimal alignment*) between s and t under a scoring function $\omega{:}\bar{\Sigma} \times \bar{\Sigma} \to \mathbb{R}$, where $\bar{\Sigma} = \Sigma \cup \{-\}$ [12] [2]. With all the previous definitions in mind, the Chain Alignment Problem can be formally stated as follows:

[1] In this paper, we will only consider alphabets that do not include the *space* (denoted by the symbol $-$) as one of its elements.

[2] For the sake of simplicity, we will also use the terms *similarity* and *optimal alignment* with chains. This means that when we refer to the similarity (optimal alignment) between two chains $\Gamma_{\mathcal{B}}$ and $\Gamma_{\mathcal{C}}$, we are referring to the similarity (optimal alignment) between $\Gamma^{\bullet}_{\mathcal{B}}$ and $\Gamma^{\bullet}_{\mathcal{C}}$.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

Chain Alignment Problem (CAP): given two strings s and t, an ordered set of segments $\mathcal{B} = \{b_1, b_2, \ldots, b_u\}$ of s, an ordered set of segments $\mathcal{C} = \{c_1, c_2, \ldots, c_v\}$ of t, and a scoring function ω, find a chain $\Gamma_{\mathcal{B}} = \{b_p, b_q, \ldots, b_r\}$ of $\mathcal{B}$ and a chain $\Gamma_{\mathcal{C}} = \{c_w, c_x, \ldots, c_y\}$ of $\mathcal{C}$ such that $sim_\omega(\Gamma^\bullet_{\mathcal{B}}, \Gamma^\bullet_{\mathcal{C}})$ is maximum among all chains of $\mathcal{B}$ and $\mathcal{C}$.

Figure 1 illustrates an instance of the CAP.

Fig. 1. An instance of the CAP and its solution. In this example we are considering the scoring function $\omega(a, b) = \{1, \text{if } a = b; -1, \text{if } a \neq b; -2, \text{if } a = - \text{ or } b = -\}$. In the center of the figure there is depicted an optimal solution to this example, the chains $\Gamma_{\mathcal{B}}$ and $\Gamma_{\mathcal{C}}$, and an optimal alignment between $\Gamma^\bullet_{\mathcal{B}}$ and $\Gamma^\bullet_{\mathcal{C}}$.

The **Multiple Chain Alignment Problem**, in turn, is a multiple version of the CAP and can be defined as follows:

Multiple Chain Alignment Problem (MCAP): given $n > 2$ strings $s_1, s_2, \ldots, s_n$, n ordered sets of segments $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$, where $\mathcal{B}_i$ is an ordered set of segments of s_i, and a scoring function ω, find n chains $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$, where Γ_i is a chain of $\mathcal{B}_i$, such that $\sum_{i=2}^{n} \sum_{j=1}^{i-1} sim_\omega(\Gamma_i^\bullet, \Gamma_j^\bullet)$ is maximum.

3 An algorithmic solution for the CAP

It is easy to check that a brute-force algorithm is not suitable to solve the CAP. However, a careful analysis of this problem shows that it exhibits the *optimal substructure* and *overlapping subproblems* properties, and as such, it is possible to develop an efficient algorithm for it using dynamic programming.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

In order to understand the recurrence that solves the CAP, consider the following definitions. Let $b_k[1..i] = b_k[1] \bullet b_k[2] \bullet \ldots \bullet b_k[i]$ and let $\Gamma_{\mathcal{B}}(k[i]) = \{b_q, b_r, \ldots, b_k[1..i]\}$ be a chain of an ordered set of segments $\mathcal{B}$ ending with $b_k[1..i]$. For the sake of simplicity, we will sometimes denote $\Gamma_{\mathcal{B}}(k[|b_k|])$ simply by $\Gamma_{\mathcal{B}}(k)$. Given an instance $I = \langle s, t, \mathcal{B} = \{b_1, \ldots, b_u\}, \mathcal{C} = \{c_1, \ldots, c_v\}, \omega\rangle$ of the CAP, let $\Gamma^{\star}_{k[i]l[j]}$ be a pair of chains $(\Gamma^{\star}_{\mathcal{B}}(k[i]), \Gamma^{\star}_{\mathcal{C}}(l[j]))$, where $\Gamma^{\star}_{\mathcal{B}}(k[i])$ is a chain of $\mathcal{B}$ ending with $b_k[1..i]$ and $\Gamma^{\star}_{\mathcal{C}}(l[j])$ is a chain of $\mathcal{C}$ ending with $c_l[1..j]$ such that $v(\Gamma^{\star}_{k[i]l[j]}) = sim_{\omega}(\Gamma^{\star}_{\mathcal{B}}(k[i])^{\bullet}, \Gamma^{\star}_{\mathcal{C}}(l[j])^{\bullet})$ is maximum, i.e., $sim_{\omega}(\Gamma^{\star}_{\mathcal{B}}(k[i])^{\bullet}, \Gamma^{\star}_{\mathcal{C}}(l[j])^{\bullet}) \geq sim_{\omega}(\Gamma_{\mathcal{B}}(k[i])^{\bullet}, \Gamma_{\mathcal{C}}(l[j])^{\bullet})$ for all pairs of chains $\Gamma_{\mathcal{B}}(k[i])$ of $\mathcal{B}$ ending with $b_k[1..i]$ and $\Gamma_{\mathcal{C}}(l[j])$ of $\mathcal{C}$ ending with $c_l[1..j]$. By definition, $\Gamma^{\star}_{k[i]l[j]}$ is defined for all $1 \leq k \leq |\mathcal{B}|$, $1 \leq l \leq |\mathcal{C}|$, $1 \leq i \leq |b_k|$, and $1 \leq j \leq |c_l|$. Lastly, let $\mathcal{M}$ be a four-dimensional matrix such that

$$\mathcal{M}[i,j,k,l] = v(\Gamma^{\star}_{k[i]l[j]}) = \max_{\substack{\text{all pairs of chains} \\ (\Gamma_{\mathcal{B}}(k[i]), \Gamma_{\mathcal{C}}(l[j]))}} sim_{\omega}(\Gamma_{\mathcal{B}}(k[i])^{\bullet}, \Gamma_{\mathcal{C}}(l[j])^{\bullet}).$$

As defined, $\mathcal{M}$ can be efficiently calculated by means of a recurrence using dynamic programming.

For a better understanding of the base cases of this recurrence, consider that the sets of segments $\mathcal{B}$ and $\mathcal{C}$ will each include a *virtual segment* (a segment not given in the input) b_0 and c_0, respectively, such that $b_0 = c_0 = \epsilon$ (both are empty strings), $b_0 \prec b$ for all $b \in \mathcal{B}$ and $c_0 \prec c$ for all $c \in \mathcal{C}$. We thus define $\Gamma^{\star}_{\mathcal{B}}(0[0])$ and $\Gamma^{\star}_{\mathcal{C}}(0[0])$ as empty chains. Finally, $\mathcal{M}$ can be efficiently calculated by means of Recurrence 1 if $k = 0$ or $l = 0$ (base cases) and by means of Recurrence 2 otherwise[3].

As to Recurrence 1, its first line corresponds to the score of an optimal alignment between the empty chains $\Gamma^{\star}_{\mathcal{B}}(0[0])$ and $\Gamma^{\star}_{\mathcal{C}}(0[0])$. Its second line, to the score of an optimal alignment between a chain of $\mathcal{B}$ ending with $b_k[1..i]$ and the empty chain $\Gamma^{\star}_{\mathcal{C}}(0[0])$ of $\mathcal{C}$. The best alignment we can get here is aligning $b_k[1..i]$ with i spaces. The third line is analogous to the second one.

The general idea behind Recurrence 2 is to consider all possible extensions of previously computed subproblem solutions and choose the best one. The options the recurrence considers for this extension, called the *set of candidate alignments*, and how this extension is carried out, change according to the position of the matrix being calculated. When calculating $\mathcal{M}[i > 1, j > 1, k, l]$, i.e., finding $v(\Gamma^{\star}_{k[i]l[j]}) = sim_{\omega}(\Gamma^{\star}_{\mathcal{B}}(k[i])^{\bullet}, \Gamma^{\star}_{\mathcal{C}}(l[j])^{\bullet})$, the recurrence can extend the optimal alignment between $\Gamma^{\star}_{\mathcal{B}}(k[i-1])$ and $\Gamma^{\star}_{\mathcal{C}}(l[j-1])$ (stored in $\mathcal{M}[i-1, j-1, k, l]$) by adding $b_k[i]$ paired with $c_l[j]$ to this alignment. It also considers the options of extending the optimal alignment between $\Gamma^{\star}_{\mathcal{B}}(k[i-1])$ and $\Gamma^{\star}_{\mathcal{C}}(l[j])$ (stored in $\mathcal{M}[i-1, j, k, l]$) by adding $b_k[i]$ paired with a space and the optimal alignment between $\Gamma^{\star}_{\mathcal{B}}(k[i])$ and $\Gamma^{\star}_{\mathcal{C}}(l[j-1])$ (stored in $\mathcal{M}[i, j-1, k, l]$) by adding a space

[3] To simplify the calculation of the base cases, we are assuming in Recurrence 1 that $\omega(a, b) < 0$, if $a = -$ or $b = -$, where ω is the scoring function given in the input. Nonetheless, it can be easily modified to cope with any scoring function.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

paired with $c_l[j]$. Evidently, in this case, the set of candidate alignments consists of $\mathcal{M}[i-1, j-1, k, l]$, $\mathcal{M}[i-1, j, k, l]$, and $\mathcal{M}[i, j-1, k, l]$. Given these three options, it chooses to extend the one that maximizes the score of the optimal alignment being calculated.

$$\begin{aligned}
&\mathcal{M}[0,0,0,0] = 0;\\
&\mathcal{M}[i,0,k,0] = \sum_{t=1}^{i} \omega(b_k[t], -), \text{ for } 1 \leq i \leq |b_k| \text{ and } 1 \leq k \leq |\mathcal{B}|;\\
&\mathcal{M}[0,j,0,l] = \sum_{t=1}^{j} \omega(-, c_l[t]), \text{ for } 1 \leq j \leq |c_l| \text{ and } 1 \leq l \leq |\mathcal{C}|.
\end{aligned} \tag{1}$$

$$\mathcal{M}[i,j,k,l] = \begin{cases}
\text{if } i > 1 \text{ and } j > 1\text{:} \\
\quad \max \begin{cases} \mathcal{M}[i-1, j-1, k, l] + \omega(b_k[i], c_l[j]), \\ \mathcal{M}[i, j-1, k, l] + \omega(-, c_l[j]), \\ \mathcal{M}[i-1, j, k, l] + \omega(b_k[i], -) \end{cases} \\
\text{if } i = 1 \text{ and } j = 1\text{:} \\
\quad \max \begin{cases} \max_{b_{k'} \prec b_k,\ c_{l'} \prec c_l} \{\mathcal{M}[|b_{k'}|, |c_{l'}|, k', l'] \\ \qquad\qquad + \omega(b_k[1], c_l[1])\}, \\ \max_{c_{l'} \prec c_l} \{\mathcal{M}[1, |c_{l'}|, k, l'] + \omega(-, c_l[1])\}, \\ \max_{b_{k'} \prec b_k} \{\mathcal{M}[|b_{k'}|, 1, k', l] + \omega(b_k[1], -)\} \end{cases} \\
\text{if } i = 1 \text{ and } j > 1\text{:} \\
\quad \max \begin{cases} \mathcal{M}[1, j-1, k, l] + \omega(-, c_l[j]), \\ \max_{b_{k'} \prec b_k} \begin{cases} \mathcal{M}[|b_{k'}|, j-1, k', l] + \omega(b_k[1], c_l[j]), \\ \mathcal{M}[|b_{k'}|, j, k', l] + \omega(b_k[1], -) \end{cases} \end{cases} \\
\text{if } i > 1 \text{ and } j = 1\text{:} \\
\quad \max \begin{cases} \mathcal{M}[i-1, 1, k, l] + \omega(b_k[i], -), \\ \max_{c_{l'} \prec c_l} \begin{cases} \mathcal{M}[i-1, |c_{l'}|, k, l'] + \omega(b_k[i], c_l[1]), \\ \mathcal{M}[i, |c_{l'}|, k, l'] + \omega(-, c_l[1]) \end{cases} \end{cases}
\end{cases} \tag{2}$$

The same idea explained in the previous paragraph is used to calculate all the remaining cells of $\mathcal{M}$, changing only the set of candidate alignments. When calculating $\mathcal{M}[i = 1, j = 1, k, l]$, for instance, the recurrence takes into account more candidates. Firstly, it considers all previously computed optimal alignments ending with any segment $b_{k'} \prec b_k$ ($\Gamma^\star_{\mathcal{B}}(k')$) and any segment $c_{l'} \prec c_l$ ($\Gamma^\star_{\mathcal{C}}(l')$), which are all included in the set $\{\mathcal{M}[|b_{k'}|, |c_{l'}|, k', l']\}$, and extends them by adding $b_k[1]$ paired with $c_l[1]$. It also considers extending all optimal alignments between $\Gamma^\star_{\mathcal{B}}(k[1])$ and $\Gamma^\star_{\mathcal{C}}(l')$, $c_{l'} \prec c_l$ ($\{\mathcal{M}[1, |c_{l'}|, k, l']\}$), by adding a space paired with $c_l[1]$ to them. Analogously, it further considers extending all optimal alignments between $\Gamma^\star_{\mathcal{B}}(k')$, $b_{k'} \prec b_k$, and $\Gamma^\star_{\mathcal{C}}(l[1])$ ($\{\mathcal{M}[|b_{k'}|, 1, k', l]\}$) by adding $b_k[1]$ paired with a space to them. Finally, it chooses the best extension of all of these.

In the third case, Recurrence 2 calculates $\mathcal{M}[i = 1, j > 1, k, l]$. First off, it considers extending the optimal alignment between $\Gamma^\star_{\mathcal{B}}(k[1])$ and $\Gamma^\star_{\mathcal{C}}(l[j-1])$

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

(stored in $\mathcal{M}[1, j-1, k, l]$) by adding a space paired with $c_l[j]$ to this alignment. But it also has to look at several other candidate alignments. It further considers extending all optimal alignments between $\Gamma^\star_{\mathcal{B}}(k')$, $b_{k'} \prec b_k$, and $\Gamma^\star_{\mathcal{C}}(l[j-1])$ ($\{\mathcal{M}[|b_{k'}|, j-1, k', l]\}$) by adding $b_k[1]$ paired with $c_l[j]$ to them. The last set of candidates consists of all optimal alignments between $\Gamma^\star_{\mathcal{B}}(k')$, $b_{k'} \prec b_k$, and $\Gamma^\star_{\mathcal{C}}(l[j])$ ($\{\mathcal{M}[|b_{k'}|, j, k', l]\}$). The recurrence extends them by adding $b_k[1]$ paired with a space. After computing all these extensions, it chooses the best one. The fourth case is analogous to the third one.

Figure 2 shows a way of viewing $\mathcal{M}$ as a $(u+1) \times (v+1)$ matrix, where each cell of $\mathcal{M}$ is an *alignment submatrix* representing the best alignment ending with two specific segments.

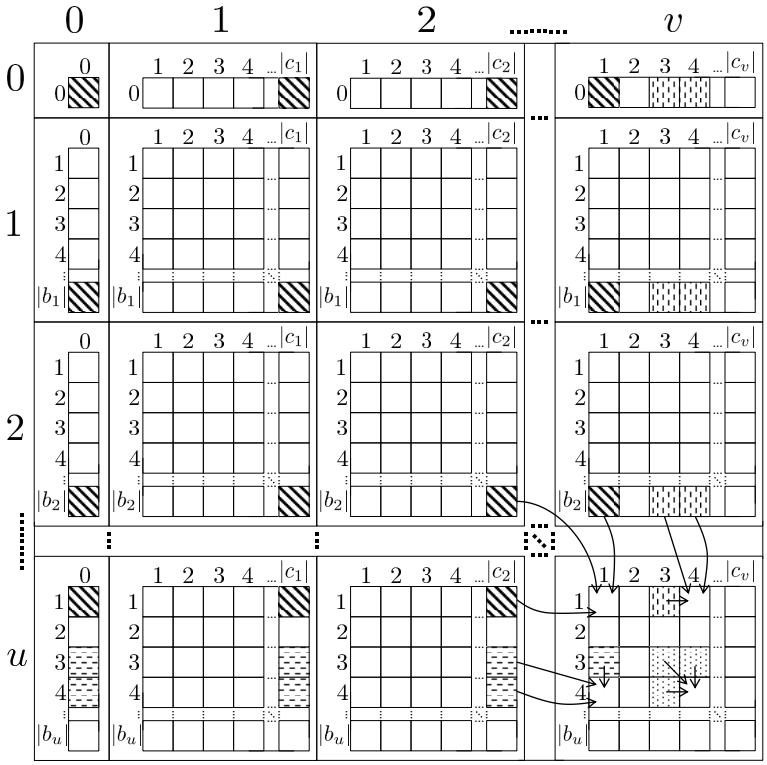

Fig. 2. A way of viewing $\mathcal{M}$ and the sets of candidate alignments considered when Recurrence 2 calculates the positions $\mathcal{M}[i > 1, j > 1, u, v]$, $\mathcal{M}[i = 1, j = 1, u, v]$, $\mathcal{M}[i = 1, j > 1, u, v]$ and $\mathcal{M}[i > 1, j = 1, u, v]$. Some arrows were hidden for the purpose of clarity, but cells with the same hatching pattern are in the same set of candidate alignments. As we can see in this specific example, it is necessary that $b_0 \prec b_1 \prec \ldots \prec b_u$ and $c_0 \prec c_1 \prec \ldots \prec c_v$ (i.e., no overlapping segments in $\mathcal{B}$ and $\mathcal{C}$), for the sets of candidate alignments to be like those depicted above.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

Upon completion of $\mathcal{M}$, the value of the optimal solution can be found as

$$\max_{b_k \in \mathcal{B}, c_l \in \mathcal{C}} \{\mathcal{M}[|b_k|, |c_l|, k, l]\}.$$

The solution for the CAP, i.e., a chain $\Gamma_{\mathcal{B}}$ of $\mathcal{B}$ and a chain $\Gamma_{\mathcal{C}}$ of $\mathcal{C}$ such that $sim_\omega(\Gamma^\bullet_{\mathcal{B}}, \Gamma^\bullet_{\mathcal{C}})$ is maximum, can be constructed by a traceback procedure. This procedure builds a path that begins at the position in $\mathcal{M}$ that stores the value of the optimal solution, and goes back to the cell used to calculate this value. It keeps building this path backwards until $\mathcal{M}[0,0,0,0]$ is reached. To determine $\Gamma_{\mathcal{B}}$ ($\Gamma_{\mathcal{C}}$), the procedure logs every segment of $\mathcal{B}$ ($\mathcal{C}$), except b_0 (c_0), in this path in LIFO order.

The proposed algorithm runs in $\mathcal{O}(|\mathcal{B}| * |\mathcal{C}| * b_{max} * c_{max})$ time and it needs $\mathcal{O}(|\mathcal{B}| * |\mathcal{C}| * b_{max} * c_{max})$ space to store $\mathcal{M}$, where b_{max} (resp. c_{max}) is the size of the longest segment in $\mathcal{B}$ (resp. $\mathcal{C}$).

4 The MCAP is $\mathcal{NP}$-hard

In this section it is shown that the MCAP is $\mathcal{NP}$-hard, that is, it has a polynomial-time algorithm if and only if $\mathcal{P} = \mathcal{NP}$. To this end, consider the following decision version of the MCAP, called MCAPD:

Multiple Chain Alignment Problem - decision version (MCAPD): given $n > 2$ strings $s_1, s_2, \ldots, s_n$, n ordered sets of segments $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$, where $\mathcal{B}_i$ is an ordered set of segments of s_i, a scoring function ω, and a positive integer l, are there n chains $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$, where Γ_i is a chain of $\mathcal{B}_i$, such that $\sum_{i=2}^{n} \sum_{j=1}^{i-1} sim_\omega(\Gamma^\bullet_i, \Gamma^\bullet_j) = l$?

To prove the $\mathcal{NP}$-completeness of the MCAPD, we will reduce from the following decision version of the Longest Common Subsequence Problem:

Longest Common Subsequence Problem - decision version (LCSPD): given $n > 2$ strings $s_1, s_2, \ldots, s_n$ and a positive integer k, is there a string t with $|t| = k$ such that t is a subsequence of $s_1, s_2, \ldots, s_n$?

A $\mathcal{NP}$-completeness proof of the LCSPD can be found in [10].

Theorem 1. *The* MCAPD *is* $\mathcal{NP}$*-complete.*

Proof. It is not hard to see that the MCAPD is in $\mathcal{NP}$. For this matter, consider a simple verification algorithm that receives as input an instance $\langle \{s_1, s_2, \ldots, s_n\}, \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n\}, \omega, l \rangle$ of the MCAPD and a certificate $\langle \Gamma_1, \Gamma_2, \ldots, \Gamma_n \rangle$, and then checks if each Γ_i is a chain of $\mathcal{B}_i$ and if $\sum_{i=2}^{n} \sum_{j=1}^{i-1} sim_\omega(\Gamma^\bullet_i, \Gamma^\bullet_j) = l$. It is trivial to see that this naive algorithm runs in polynomial time.

For the reduction step, suppose there is a polynomial-time decision algorithm for the MCAPD called *AlgMCAPD*. We will thus prove there is a polynomial-time algorithm for the LCSPD by transforming an arbitrary instance I =

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

$\langle\{s_1, s_2, ..., s_n\}, k\rangle$ of the LCSPD into an instance $I' = \langle\{s'_1, s'_2, ..., s'_n\}, \{\mathcal{B}_1, \mathcal{B}_2, ..., \mathcal{B}_n\}, \omega, l\rangle$ of the MCAPD such that I is positive if and only if I' is positive.

In what follows, four steps are described to build I' from I. In the first step, $\{s_1, s_2, ..., s_n\}$ is assigned to $\{s'_1, s'_2, ..., s'_n\}$, i.e., the sets of strings are the same for both instances.

The second step defines the n ordered sets of segments $\mathcal{B}_1, \mathcal{B}_2, ..., \mathcal{B}_n$. An ordered set of segments $\mathcal{B}_i$ is constructed by adding, in order, each symbol of s_i as a segment of $\mathcal{B}_i$.

The third step defines the scoring function ω as

$$\omega(a,b) = \begin{cases} 1, & \text{if } a = b \\ -(Opt^+ + 1), & \text{if } a \neq b \\ -(Opt^+ + 1), & \text{if } a = - \text{ or } b = -, \end{cases}$$

where $Opt^+ = \frac{n*(n-1)}{2} * |s_{max}|$ and s_{max} is the longest input string. In other words, Opt^+ is an upper bound for $\sum_{i=2}^{n} \sum_{j=1}^{i-1} sim_\omega(\Gamma_i^\bullet, \Gamma_j^\bullet)$.

Finally, the fourth step defines

$$l = \frac{n * (n-1) * k}{2}.$$

It is not hard to check that the transformation of I into I', using the four steps previously explained, takes polynomial time. It is thus possible to build an algorithm, called *AlgLCSPD*, to decide the LCSPD. This algorithm receives an arbitrary instance I of the LCSPD, transforms I into an instance I' of the MCAPD, and then calls *AlgMCAPD* taking I' as argument. *AlgLCSPD* then decides I as positive (negative) if *AlgMCAPD* decides I' as positive (negative). Since both the transformation of I into I' and the execution of *AlgMCAPD* take polynomial time, it is easy to see that *AlgLCSPD* also runs in polynomial time. To complete the proof, we need to show that I is positive if and only if I' is positive.

(Proof of "only if.") Suppose I is positive. In this case, there is a string t with $|t| = k$ such that t is a subsequence of $s_1, s_2, \ldots, s_n$. We will prove that I' is also positive. As I' is defined, it is always possible to build n chains $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$ such that $\Gamma_1^\bullet = \Gamma_2^\bullet = ... = \Gamma_n^\bullet = t$. Therefore,

$$\begin{aligned} \sum_{i=2}^{n} \sum_{j=1}^{i-1} sim_\omega(\Gamma_i^\bullet, \Gamma_j^\bullet) &= \sum_{i=2}^{n} \sum_{j=1}^{i-1} sim_\omega(t,t) \\ &= \sum_{i=2}^{n} \sum_{j=1}^{i-1} |t| \\ &= \sum_{i=2}^{n} \sum_{j=1}^{i-1} k \\ &= \frac{n*(n-1)*k}{2} \\ &= l. \end{aligned}$$

Hence I' is positive.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

(Proof of "if.") Suppose I' is positive. Thus, there are n chains $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$, where Γ_i is a chain of $\mathcal{B}_i$, such that $\sum_{i=2}^{n}\sum_{j=1}^{i-1} sim_\omega(\Gamma_i^\bullet, \Gamma_j^\bullet) = l$. We will show that I is also positive.

Since I' is positive, we can show that $\Gamma_1 = \Gamma_2 = ... = \Gamma_n$. Suppose, by contradiction, that $\Gamma_1 = \Gamma_2 = ... = \Gamma_n$ is not true, i.e., there is at least one pair of chains (Γ_i, Γ_j) such that $\Gamma_i \neq \Gamma_j$. In this case, in at least one of the $\frac{n*(n-1)}{2}$ alignments between $\Gamma_1^\bullet, \Gamma_2^\bullet, ..., \Gamma_n^\bullet$, there will be a mismatch or a space. As the score of a mismatch or a space is negative enough to make $\sum_{i=2}^{n}\sum_{j=1}^{i-1} sim_\omega(\Gamma_i^\bullet, \Gamma_j^\bullet) < 0$, we have a contradiction of our hypothesis that $\sum_{i=2}^{n}\sum_{j=1}^{i-1} sim_\omega(\Gamma_i^\bullet, \Gamma_j^\bullet) = l$, since l is a positive integer.

As $\Gamma_1 = \Gamma_2 = ... = \Gamma_n$, we have that $\Gamma_1^\bullet = \Gamma_2^\bullet = ... = \Gamma_n^\bullet$. Besides, due to the way we transformed I into I', it is easy to check that each $\Gamma_i^\bullet$ is a subsequence of $s_1, s_2, ...s_n$. With the previous observations in mind, we have

$$\begin{aligned}\sum_{i=2}^{n}\sum_{j=1}^{i-1} sim_\omega(\Gamma_i^\bullet, \Gamma_j^\bullet) &= \sum_{i=2}^{n}\sum_{j=1}^{i-1} sim_\omega(\Gamma_x^\bullet, \Gamma_x^\bullet) \\ &= \sum_{i=2}^{n}\sum_{j=1}^{i-1} |\Gamma_x^\bullet| \\ &= \frac{n*(n-1)*|\Gamma_x^\bullet|}{2}.\end{aligned}$$

Since I' is positive, $\sum_{i=2}^{n}\sum_{j=1}^{i-1} sim_\omega(\Gamma_i^\bullet, \Gamma_j^\bullet) = l$. Thus, we have

$$l = \frac{n*(n-1)*|\Gamma_x^\bullet|}{2}$$

$$|\Gamma_x^\bullet| = \frac{2*l}{n*(n-1)} = k.$$

Therefore, if $t = \Gamma_x^\bullet$, for all $1 \leq x \leq n$, we have that $|t| = k$ so that t is a subsequence of $s_1, s_2, \ldots, s_n$. Hence, I is positive.

■

5 Heuristics for the Multiple Chain Alignment Problem

Given the $\mathcal{NP}$-hardness of the MCAP, we propose here three heuristics for it. Two of them are based on the solution proposed by us for the CAP and the third heuristic is based on the algorithm described in [4] to solve the Spliced Alignment Problem (SAP). Informally, this problem consists in, given as input two strings s and t, and an ordered set of segments $\mathcal{B} = \{b_1, b_2, \ldots, b_u\}$ of s, finding an ordered subset $\Gamma = \{b_p, b_q, \ldots, b_r\}$ of non-overlapping segments of $\mathcal{B}$ such that the string resulting from the concatenation of all segments in Γ is very similar to t.

For a better understanding of the proposed heuristics, consider the following. The input to the Multiple Chain Alignment Problem is given by the triple $(\{s_1, s_2, \ldots, s_n\}, \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n\}, \omega)$. We can invoke the algorithm that solves the Chain Alignment Problem as $CAP(s_i, s_j, \mathcal{B}_i, \mathcal{B}_j, \omega)$, which in turn

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

will return a pair of chains $\mathcal{P}_{i,j} = (\Gamma_i, \Gamma_j)$ such that $sim_\omega(\Gamma_i^\bullet, \Gamma_j^\bullet)$ is maximum among all chains of $\mathcal{B}_i$ and $\mathcal{B}_j$. Further, given a pair of chains $\mathcal{P}_{i,j}$, we can reference its first chain by $\mathcal{P}_{i,j}.\Gamma_i$, its second chain by $\mathcal{P}_{i,j}.\Gamma_j$, and its value by $val(\mathcal{P}_{i,j}) = sim_\omega(\mathcal{P}_{i,j}.\Gamma_i^\bullet, \mathcal{P}_{i,j}.\Gamma_j^\bullet)$.

All three heuristics here described work on a common framework. They compute, at each iteration i, the chain Γ_i to be included in the final solution. Thus, it is enough to describe how only one Γ_i is determined. In addition, their first step in order to calculate Γ_i is to always invoke $\mathcal{P}_{i,j} = CAP(s_i, s_j, \mathcal{B}_i, \mathcal{B}_j, \omega)$ for all $1 \leq j \leq n, j \neq i$. Thus, in the description of the heuristics, we will always suppose that all pairs $\mathcal{P}_{i,j}$ are already precomputed.

5.1 $\mathcal{H}1$ - Heuristic of the Consensus Chain

The consensus chain $\mathcal{C}(\Gamma_i)$ of an ordered set of segments $\mathcal{B}_i$ of a sequence s_i is obtained as follows. We count how many times each segment of $\mathcal{B}_i$ is present in each pair $\mathcal{P}_{i,j}$ for $1 \leq j \leq n, j \neq i$. $\mathcal{C}(\Gamma_i)$ is then defined as the segments of $\mathcal{B}_i$ that are present more than $\lfloor \frac{n-1}{2} \rfloor$ times. This heuristic thus defines Γ_i as $\mathcal{C}(\Gamma_i)$.

5.2 $\mathcal{H}2$ - Greedy Heuristic

The Greedy Heuristic determines each Γ_i as

$$\Gamma_i = \mathcal{P}_{i,j}.\Gamma_i | val(\mathcal{P}_{i,j}) = \max_{1 \leq j \leq n, j \neq i} \{val(\mathcal{P}_{i,j})\}.$$

5.3 $\mathcal{H}3$ - Heuristic of the Central Chain

For the Heuristic of the Central Chain, consider that we can invoke the algorithm described in [4] to solve the Spliced Alignment Problem as $SAP(s, t, \mathcal{B}, \omega)$, and that this invocation returns the value of optimal solution. This heuristic let

$$\Gamma_i = \mathcal{P}_{i,j}.\Gamma_i | \sum_{k=1,\ k \neq i}^{n} SAP(s_k, \mathcal{P}_{i,j}.\Gamma_i^\bullet, \mathcal{B}_k, \omega) \text{ is maximum } \forall\ 1 \leq j \leq n, j \neq i.$$

6 Application to the Gene Identification Problem

In this section we model a variant of the Gene Identification Problem as the CAP and the MCAP, and present some results which attest one of the applications of these latter problems, justifying their previous formalization and study.

In short, the Gene Identification Problem (GIP) consists in, given a DNA sequence, determine the exons of the genes encoded by the given sequence [11]. The GIP is one of the most important problems in Molecular Biology, since it is directly related to disease prevention and treatment, pest control, etc. However, even nowadays, it still remains a challenging and complicated problem, which justifies additional research on this topic.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

We propose to tackle a variant of the GIP by modelling it as the CAP. To formally define this variant, we briefly recall the definition of some biological concepts. Firstly, consider the fact that an organism's DNA may change permanently during time. These changes are called *mutations* and they can be either beneficial, neutral or negative, according to their effect on the organism. It is also important to note that mutations are observed more frequently on non-coding regions of the DNA, which impact less on the protein biosynthesis process, than on coding regions. Beneficial mutations may be transferred to the organism's descendants and, in some cases, even originate new species. In this context, given two different organisms $\mathcal{O}_1$ and $\mathcal{O}_2$, we say that a gene $\mathcal{G}_1$ of $\mathcal{O}_1$ is homologous to a gene $\mathcal{G}_2$ of $\mathcal{O}_2$ if $\mathcal{G}_1$ and $\mathcal{G}_2$ evolved from a same gene present in a common ancestor of $\mathcal{O}_1$ and $\mathcal{O}_2$. Finally, we can define the Homologous Gene Identification Problem as follows.

Homologous Gene Identification Problem (HGIP): given two DNA sequences $\mathcal{D}_1$ and $\mathcal{D}_2$, which contain two unknown homologous genes $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively, a set of putative exons $\mathcal{E}_1$ of $\mathcal{G}_1$, and a set of putative exons $\mathcal{E}_2$ of $\mathcal{G}_2$, find a set $\mathcal{R}_1$ of $\mathcal{E}_1$ and a set $\mathcal{R}_2$ of $\mathcal{E}_2$ such that $\mathcal{R}_1$ contains the real exons of $\mathcal{G}_1$ and $\mathcal{R}_2$ contains the real exons of $\mathcal{G}_2$.

To model the HGIP as the CAP, let's consider $\Sigma = \{A, C, G, T\}$, map the input of the HGIP to an input of the CAP, and the output of the CAP to an output of the HGIP. To map the inputs, let s, t, $\mathcal{B}$ and $\mathcal{C}$ be $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively, and let $\omega(a, b) = \{1, \text{if } a = b; -1, \text{if } a \neq b; -2, \text{if } a = - \text{ or } b = -\}$.

To map the outputs, let $\mathcal{R}_1$ and $\mathcal{R}_2$ be $\Gamma_{\mathcal{B}}$ and $\Gamma_{\mathcal{C}}$, respectively. The main supposition behind this modeling is that the coding regions of the two unknown homologous genes $\mathcal{G}_1$ and $\mathcal{G}_2$ will be similar, even if they have been affected by independent mutations. In this sense, the probability that the segments in $\Gamma_{\mathcal{B}}$ and $\Gamma_{\mathcal{C}}$ correspond to the real exons of $\mathcal{G}_1$ and $\mathcal{G}_2$ is high.

Similarly as we extended the CAP to the MCAP, we can extend the HGIP to a multiple version, where we receive n DNA sequences containing n unknown homologous genes, and a set of putative exons of each gene, and we aim to find the set of real exons of each gene. In this way, it is also possible to apply the MCAP to the HGIP. The strength of doing so stands out when several homologous genes are available, instead of only two. In these situations, we apply the heuristics proposed for the MCAP to make an effective use of this additional data, in order to get better solutions.

The dataset considered to evaluate the accuracy of the proposed model was built using data from the ENCODE project [3] and from the HomoloGene database [15]. In short, the ENCODE project aimed at identifying all functional elements in the human genome sequence, and the HomoloGene database allowed for the automatic detection of homologous genes from several eukaryotic, completely sequenced DNAs. To build the dataset, we firstly selected all genes from the 44 regions studied in the pilot phase of ENCODE project, such that 1) coded for only one protein; 2) their size were a multiple of 3; 3) presented the canonical exon-intron structure. For each of these selected human genes, we searched

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

for homologous genes from other species by using HomoloGene. We removed all homologous sequences that: 1) did not code for a protein; 2) coded for hypothetical proteins; 3) not all exons were completely identified; 4) did not present the canonical exon-intron structure; 5) had exons composed by other bases besides A, C, G or T. Finally, we added to the dataset the DNA sequences encoding all genes selected so far (these sequences also included 1000 bases before the start of the gene and after its end).

To finish the construction of the dataset, a simple tool was implemented to retrieve the set of putative exons for each DNA sequence. To do so, we used the method GAP3 described in [6] coupled with some metrics. For each pair of human and homologous sequences, GAP3 is applied to find the similar regions of both sequences, which we consider as the putative exons. As we might have a large number of putative exons in this step, we decided to filter the 50 best, by using a combination of four metrics: the WAMs defined in [14], the codon usage [5], the optimal general alignment score [6], and the PWMs defined in [2]. We would like to note that the sequences in the dataset presented an average of 6.8 exons. However, the sets of putative exons generated by this tool included less than 50% of the real exons. Thus, we ultimately decided to add the missing real exons to each set. Finally, the dataset included 206 test instances.

The results of the application of the three heuristics for the MCAP and the algorithm that solves the CAP to the HGIP can be found in Table 1. The accuracy of each solution was evaluated by using the measures introduced by Burset and Guigó in [1] to evaluate the performance of gene prediction tools on three distinct levels: nucleotide, exon and border. Briefly speaking, at the nucleotide level, a true positive (TP) is a coding nucleotide that was correctly predicted as coding, a true negative (TN) is a non-coding nucleotide that was correctly predicted as non-coding, a false positive (FP) is a non-coding nucleotide that was incorrectly predicted as coding, and a false negative (FN) is a coding nucleotide that was incorrectly predicted as non-coding. Considering this, the specificity (resp. sensitivity) at the nucleotide level is defined as $Sp_n = \frac{TP}{TP+FP}$ (resp. $Sn_n = \frac{TP}{TP+FN}$), and the approximate correlation, which summarize both measures, as $Ac = \frac{1}{2} * (\frac{TP}{TP+FN} + \frac{TP}{TP+FP} + \frac{TN}{TN+FP} + \frac{TN}{TN+FN}) - 1$. At the exon level, let $\mathcal{R}_e$ be the set of real exons and let $\mathcal{P}_e$ be the set of predicted exons. The specificity (resp. sensitivity) at the exon level is thus defined as $Sp_e = \frac{|\mathcal{R}_e \bigcap \mathcal{P}_e|}{|\mathcal{P}_e|}$ (resp. $Sn_e = \frac{|\mathcal{R}_e \bigcap \mathcal{P}_e|}{|\mathcal{R}_e|}$) and both measures are summarized as $Av_e = \frac{Sp_e+Sn_e}{2}$. For the border level, suppose that the exon borders denote its start and end positions, and let $\mathcal{PR}_b$ be the set of exon borders that were correctly predicted, let $\mathcal{R}_e$ be the set of real exons, and let $\mathcal{P}_e$ be the set of predicted exons. The specificity (resp. sensitivity) at the border level is thus defined as $Sp_b = \frac{|\mathcal{PR}_b|}{2*|\mathcal{P}_e|}$ (resp. $Sn_b = \frac{|\mathcal{PR}_b|}{2*|\mathcal{R}_e|}$) and both measures are summarized as $Av_b = \frac{Sp_b+Sn_b}{2}$.

In this work these measures were calculated considering only the predictions on the human sequence. The algorithm for the CAP was applied to the human sequence and to each of its homologous, and the best solution was selected.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

Table 1. The results of the application of the three heuristics for the MCAP and the algorithm that solves the CAP.

Algorithm	Nucleotide			Exon			Border		
	Sp_n	Sn_n	Ac	Sp_e	Sn_e	Av_e	Sp_b	Sn_b	Av_b
$\mathcal{H}1$	**0,945**	0,935	0,923	**0,897**	0,881	**0,889**	**0,919**	0,901	0,910
$\mathcal{H}2$	0,910	**0,994**	0,940	0,803	0,880	0,841	0,838	0,925	0,882
$\mathcal{H}3$	0,942	0,988	**0,958**	0,865	**0,909**	0,887	0,894	**0,943**	**0,919**
CAP	0,916	0,969	0,933	0,826	0,881	0,854	0,856	0,916	0,886

As we can see in Table 1, the CAP got the worst performance, followed by $\mathcal{H}2$. The best results were obtained by $\mathcal{H}3$, followed by $\mathcal{H}1$, as $\mathcal{H}3$ got the best performance on Ac and Av_b, and lost on Av_e to $\mathcal{H}1$ by a very small margin. As expected, these results show that if we increase the number of evidences in the HGIP, we can get better results. In this specific test, by obtaining several homologous genes to a human gene, and by processing these data collectively, using the heuristics for the MCAP, it was possible to produce better results than processing these homologous sequences individually, using the algorithm for the CAP. We can conclude then, even though we do not have an exact solution for the MCAP, in some applications, the extra data and evidence given by multiple inputs are too valuable to be ignored. Thus, future works can focus on a deeper study of better heuristics or approximate solutions for the MCAP.

7 Concluding remarks

This paper presented two new string related combinatorial optimization problems, namely the Chain Alignment Problem and the Multiple Chain Alignment Problem, and some results for them. More specifically, the first problem was solved via a polynomial-time dynamic programming algorithm. For the second one, it was proved that it is hard to develop a polynomial-time exact algorithm, unless $\mathcal{P} = \mathcal{NP}$. We also attested that these problems can model practical problems, by applying them to a variant of the Gene Identification Problem, and by getting appropriate results.

A particularly interesting point to be investigated concerns the cases where each ordered set of segments β_i of s_i has linear size on s_i. In such cases, algorithms for the classical Spliced Alignment Problem were improved from cubic to almost quadratic time [8, 13, 17]. Surely, such restriction may not be applicable depending on the modeled problem, but regarding the main application of both the Chain Alignment Problem and the Multiple Chain Alignment Problem, namely the Gene Identification Problem, such constraint is appropriate. Therefore, the techniques described in those papers could lead to faster solutions to the problems presented in this work. It is also worth noting that the results shown in this paper for the Multiple Chain Alignment Problem assumed a specific scoring function. As future works, it would be of interest to explore the complexity of this problem under other scoring functions, especially those usu-

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2

ally employed in practical problems. Still regarding this problem, the study of probabilistic algorithms, linear programming models, and heuristics for it could reveal interesting results. The application of both problems in other practical tasks also remains as a future work.

Acknowledgements. The authors acknowledge Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brazil, for the support of this work.

References

1. Burset, M., Guigó, R.: Evaluation of gene structure prediction programs. Genomics **34**, 353–367 (1996)
2. Cavener, D.R., Ray, S.C.: Eukaryotic start and stop translation sites. Nucleic Acids Research **19**(12), 3185–92 (1991)
3. Consortium, T.E.P.: The ENCODE (ENCyclopedia Of DNA Elements) Project. Science **306**(5696), 636–640 (2004)
4. Gelfand, M.S., Mironov, A.A., Pevzner, P.A.: Gene Recognition Via Spliced Sequence Alignment. Proceedings of the National Academy of Sciences of the United States of America **93**, 9061–9066 (1996)
5. Guigo, R.: GENETIC DATABASES: DNA Composition, Codon Usage and Exon Prediction, chap. DNA Composition, Codon Usage and Exon Prediction (chapter 4), pp. 53–80. Academic Press (1999)
6. Huang, X., Chao, K.M.: A generalized global alignment algorithm. Bioinformatics **19**(2), 228–233 (2003)
7. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice-Hall. (1988)
8. Kent, C., Landau, G.M., Ziv-Ukelson, M.: On the complexity of sparse exon assembly. Journal of Computational Biology **13**(5), 1013–1027 (2006)
9. Lewin, B., Krebs, J., Goldstein, E., Kilpatrick, S.: Lewin's Genes X. Jones and Bartlett (2009)
10. Maier, D.: The Complexity of Some Problems on Subsequences and Supersequences. J. ACM **25**(2), 322–336 (1978)
11. Majoros, W.: Methods for computational gene prediction. Cambridge University Press (2007)
12. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology **48**, 443–453 (1970)
13. Sakai, Y.: An almost quadratic time algorithm for sparse spliced alignment. Theory of Computing Systems **48**(1), 189–210 (2011)
14. Salzberg, S.: A method for identifying splice sites and translational start sites in eukaryotic mRNA. Bioinformatics **13**(4), 365–376 (1997)
15. Sayers, E.W., et al.: Database resources of the National Center for Biotechnology Information. Nucleic Acids Research **38**(Database-Issue), 5–16 (2010)
16. Sebastiani, F.: Machine learning in automated text categorization. ACM Computing Surveys **34**(1), 1–47 (2002)
17. Tiskin, A.: Semi-local string comparison: Algorithmic techniques and applications. Mathematics in Computer Science **1**(4), 571–603 (2008)
18. Ye, Y., Jaroszewski, L., Li, W., Godzik, A.: A segment alignment approach to protein comparison. Bioinformatics **19**, 742–749 (2003)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_2