On the mathematical characterization of sequence alignments

MSc thesis

Author:

Kristóf Takács

Applied mathematics MSc

Supervisor:

Dr. Vince Grolmusz

professor

Department of Computer Science

Eötvös Loránd University

2016

Contents

Acknowledgement

I would like to thank my supervisor, Dr. Vince Grolmusz for his many suggestions and much help in writing this thesis. I am sure that the current version of this work could not be established without his proposals and inspirations.

I also thank my family for their support and encouragement while writing this thesis: these months, they helped me in many ways as well as in my entire life.

1. Introduction

Aligning more than two sequences with as little cost as possible is a quite essential problem for those who are interested in bioinformatical research. E.g., by this method, some information can be gained on the conservative regions of some particular sequences that can generally determine the basic functions and parameters of a group of DNA, RNA or protein sequences [3] [4]. Multiple sequence alignment (MSA) is one of the most important tools that is used during motif finding: e.g., if one is given some gene sequences so that is known they perform the same function in different species, it is a plausible question that exactly what these sequences have in common?

Considering the practical significance of the multiple sequence alignment, it is not surprising that this problem is in the centre of bioinformatical research for decades. The practical motivation of this problem indicated that most people who attended to MSA have tried to find a new result for the general question, construct a better approximation algorithm or prove something about the complexity of MSA. It was probably the most important achievement in this topic when Isaac Elias has proved that MSA is **NP**-complete if the score scheme of the characters is a metric [2]. (This negative result can be even more interesting if we consider that for two sequences the Needleman-Wunsch algorithm generates an optimal alignment in $O(n^2)$ time [6].)

Even though it is unlikely that one can find a fast and accurate general algorithm for MSA, several heuristic approximation algorithms have been developed during last decades, because of the importance of this problem. One of the most frequently used among them is named Clustal and it applies a progressive method: first, this algorithm builds a so-called "guide tree" to determine in which order it is the most practical to align the sequences, then using this tree, it creates a multiple alignment. This process starts with aligning the two closest sequences optimally, and after that, in every step, a sequence that is not aligned yet will be aligned, or two sets of aligned sequences will be aligned to each other optimally [8]. Many other heuristic algorithms for MSA are applied widely which can use progressive methods like Clustal (e.g., T-Coffee), iterative methods (DIALIGN) or even tools of probability theory (e.g., Hidden Markov models from the POA heuristic) [7].

Summing up, it is known that the MSA problem is hard in general, but it is still an interesting question that precisely when it begins to be really hard. One can assume that for length-1 (and perhaps even for length-2) sequences, it may not be that hard to find an optimal alignment. Furthermore, if an optimal alignment for short sequences can be determined in polynomial time, then it could also help to develop faster or more accurate heuristic algorithms.

In my thesis, some new results regarding the alignment of short sequences are presented. In my opinion, it can be the most relevant result of the first part of my work that for length-1 sequences using arbitrary metric, as well as for length-2 sequences using some special metric, the optimum of the MSA problem can be easily, in the most trivial way, determined. Furthermore, some new general results in aligning arbitrary length sequences are contained in my thesis; besides that, a new heuristic algorithm is also described. After a theoretical exposition of these heuristics, comparision results with the aforementioned MSA algorithm Clustal are presented. All of the three heuristics are based on the same main idea (using elements of the optimal pairwise alignments).

2. Definitions and notations

Let $\Sigma = \{a_1, \ldots, a_n\}$ be a finite alphabet. A string over Σ is called a *sequence.* s'_1 and s'_2 is an *alignment* of sequences s_1 and s_2 if $(\forall\ i)$ s'_i is obtained from s_i by inserting gaps (spaces, denoted by $-$) into or at either end of s_i and by that, s'_1 and s'_2 have the same length. (It is assumed that $-$ is not an element of Σ.) Because of this definition, every character of s'_1 is uniquely corresponded to a character of s'_2.

Let l be the common length of s'_1 and s'_2. The *cost* of this alignment is equal to $\sum_{i=1}^{l} d(s'_1(i), s'_2(i))$, where d is a *score scheme* over $\Sigma \cup \{-\}$ and $s'_j(i)$ is the ith character of s'_j. It is commonly required that a score scheme must satisfy triangle inequality: $\forall i, j, k :\ d(a_i, a_j) \leq d(a_i, a_k) + d(a_k, a_j)$. A frequently used score scheme is the so-called *unit metric*, where $d(a_i, a_j) = 0$ if $i = j$ and 1 otherwise. We call an alignment *optimal* for two sequences if its cost is minimal among every possible alignments.

The definiton of aligning two sequences can be easily generalized for more strings: let k be the number of sequences to align. Let us insert gaps into or at either end of every strings so that they have the same l length and in the proper order, write them under each other. We call this matrix size of kxl a *multiple alignment* of these sequences. There are different *scoring methods* how to define the *cost* of a multiple alignment, perhaps the most often used one is the *sum of pairs* method: using this, we get the required cost of an alignment as the sum of the costs of aligning the $\binom{k}{2}$ pairs from the aligned sequences. In a formula: if $s_1, \ldots, s_k$ are sequences to align, then their sum of pair cost is $\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} cost(s_i, s_j)$ [10].

The next definition of the MSA problem that I have worked up is containing another approach of this problem using functions.

Definition 1 *Let $S = \{s_1, \ldots, s_k\}$ be a set of sequences over Σ where $\forall i : |s_i| = n$ and denote $s_i(j)$ the j-th character of s_i. A set of injective functions $f_i : \{1, \ldots, n\} \to \{1, \ldots, N\}$ $(1 \le i \le k, n \le N \le nk)$ is a multiple sequence alignment of S if*

- $\forall 1 \le i \le k$: f_i strictly monotonically increasing and

- $\forall 1 \le j \le N$ $\exists f_i$ $\exists l : f_i(l) = j$.

Let s'_i denote an N-length string where $s'_i(f_i(j)) = s_i(j)$ and if $j \notin R(f_i)$, then $s'_i(j) = -$.

Let $d(s'_i, s'_j) = \sum_{m=1}^{N} d(s'_i(m), s'_j(m))$, where d is a distance on $\Sigma \cup \{-\}$. A multiple sequence alignment is optimal if $\sum_{i=1}^{k} \sum_{\substack{j=1 \\ j \neq i}}^{k} d(s'_i, s'_j)$ is minimal.

Examples. *i*) Let S be $S := \{CCG, GCG, CGC\}$. The following set of aligned sequences is a multiple alignment $\mathcal{A}$ of S:

$$\begin{array}{cccc} \text{C} & \text{C} & \text{G} & - \\ \text{G} & \text{C} & \text{G} & - \\ - & \text{C} & \text{G} & \text{C} \end{array}$$

Using unit metric and considering the cost of the columns, cost($\mathcal{A}$) is equal to $3 + 0 + 0 + 2 = 5$.

ii) Let Σ now contain only two characters (C and G) with the following metric:

	C	G	$-$
C	0	2	1
G	2	0	1
$-$	1	1	0

Let S be $S := \{CG, GC, GG\}$. In this case, the following set is a multiple alignment $\mathcal{A}$ of S:

$$\begin{array}{ccc} - & \text{C} & \text{G} \\ \text{G} & \text{C} & - \\ \text{G} & - & \text{G} \end{array}$$

Using the given metric, cost($\mathcal{A}$) will be equal to $2 + 2 + 2 = 6$.

3. Multiple sequence alignment for short sequences

3.1. Multiple sequence alignment for length-1 sequences

In this section, let us focus on aligning length-1 sequences (equivalently, characters of Σ). To prove a theorem regarding sequences of this kind, an important earlier result must be used:

Theorem 1 ([1]) *Let U be a subset of a set S of sequences over Σ such that U contains only identical sequences, and let $\mathcal{A}$ be an optimal alignment of S. Then* $d(\mathcal{A}_U) = \sum\limits_{u_i \in U} \sum\limits_{\substack{u_j \in U \\ i<j}} d(u_i, u_j) = 0$ *in $\mathcal{A}$.*

As an important corollary, this theorem is implying that it is enough to examine sets of sequences where these sequences are pairwise different, because in an optimal alignment, every instance of a given sequence is aligned identically.

The next definition will be used frequently throughout this chapter:

Definition 2 *Let S be a set of sequences that have the same length. $\mathcal{A}$ is the trivial alignment of S if $\mathcal{A}$ is constructed by writing every sequnce under each other, without using gaps.*

3.1.1. Multiple sequence alignment for length-1 sequences using unit metric

The main result of this subsection is the following theorem:

Theorem 2 *Using unit metric, no multiple sequence alignment for length-1 sequences exists that has less cost than their trivial alignment. Suppose we align k different sequences, then the cost of an optimal alignment is $\binom{k}{2}$.*

Proof. Considering Theorem 1, it can be assumed that the characters that need to be aligned are pairwise different because if there were some identical ones among them, then all instances of a particular character would be aligned the same way.

It is easy to see that the trivial alignment of k different characters has a cost of $\binom{k}{2}$: there are $\binom{k}{2}$ pairs among these characters and in every pair, there are two different sequences, so the cost of an aligned pair is always 1.

If we assume that this alignment is not optimal, then the length of every aligned sequence must be at least 2 in an optimal alignment, and we have to examine the cost of an alignment of this type. If this common length of aligned sequences is $l \geq 2$, then the general structure of the nxl matrix of this multiple alignment is the next: $\forall 1 \leq i \leq l$ there are k_i characters in the ith column ($\sum\limits_{i=1}^{l} k_i = k$) and they are placed so that in every row, there is only one character and $l-1$ gaps (see Figure 1).

If we focus on the first column, we can establish that its cost is $\binom{k_1}{2} + (k-k_1)k_1$, since there are k_1 different characters with cost of $\binom{k_1}{2}$, and besides that, all of the $(k-k_1)$ gaps increases the cost by one with every alphabetical character. A similiar statement is true for every column, so the cost of this alignment: $\sum_{i=1}^{l} \binom{k_i}{2} + (k-k_i)k_i = k\sum_{i=1}^{l} k_i - \frac{\sum_{i=1}^{l} k_i}{2} - \frac{\sum_{i=1}^{l} k_i^2}{2} = k^2 - \frac{k}{2} - \frac{\sum_{i=1}^{l} k_i^2}{2}$. Thus if we want to minimize the cost of this alignment then we have to maximize $\sum\limits_{i=1}^{l} k_i^2$.

By examining $(\sum\limits_{i=1}^{l} k_i)^2$, we can see that it is equal to k^2, but at the same time, it is equal to $\sum\limits_{i=1}^{l} k_i^2 + 2\sum\limits_{i=1}^{l}\sum\limits_{\substack{j=2 \\ i<j}}^{l} k_i k_j$. From this, it is clear that $\sum\limits_{i=1}^{l} k_i^2 \leq k^2$, and by that, the cost of this alignment can not be less than $\frac{k^2-k}{2} = \binom{k}{2}$. $\square$

$$\begin{array}{cccc}
a_1 & - & \dots & - \\
\dots & \dots & \dots & \dots \\
a_{k_1} & - & \dots & - \\
- & a_{k_1+1} & \dots & - \\
\dots & \dots & \dots & \dots \\
- & a_{k_2} & \dots & - \\
\dots & \dots & \dots & \dots \\
- & - & \dots & a_{k_{l-1}+1} \\
\dots & \dots & \dots & \dots \\
- & - & \dots & a_{k_l}
\end{array}$$

Figure 1. A multiple alignment for length-1 sequences on l columns.

Note. From the proof, it is also clear (by minimizing $\sum_{i=1}^{l} k_i^2$) that a multiple alignment for k different length-1 sequences can not have a cost more than $k^2 - \frac{k}{2} - \frac{k^2}{2l}$ if the length of aligned sequences is l. Since $l \leq k$, the cost can be at most $k^2 - k$ and this limit can be reached indeed: if there is only one character in every column and in every row, then the cost will be $k(k-1) = k^2 - k$.

3.1.2. Multiple sequence alignment for length-1 sequences using arbitrary metric

In this subsection, it will be shown that for length-1 sequences, we can use any metric as score scheme, the multiple sequence alignment problem still remains as easy as in case of unit metric.

Theorem 3 *Using arbitrary metric, no multiple sequence alignment for length-1 sequences exists that has less cost than their trivial alignment. Suppose we align k different sequences, then the cost of an optimal alignment is equal to* $C = \sum_{i=1}^{k} \sum_{\substack{j=2 \\ i<j}}^{k} d(a_i, a_j)$.

Proof. Because of Theorem 1, it can be assumed again that every sequence has exactly one instance in the set S of sequences to be aligned. If we consider the trivial alignment of the S, it is easy to see that its cost is equal to C. Induction for the number of the columns in a multiple sequence alignment will be used to show that any alignment can not have lower cost than C.

Let be assumed that the trivial alignment is not optimal and let $\mathcal{A}$ denote an optimal alignment. Assuming that $\mathcal{A}$ is not the trivial alignment, $\mathcal{A}$ has l columns where $l \geq 2$. It can be showed that $\mathcal{A}$ can not have exactly two columns, because in this case, trivial alignment would have a lower cost than $\mathcal{A}$ has.

Let be assumed to the contrary that $\mathcal{A}$ has exactly two columns; so there are k_1 sequences in the first column and k_2 in the second column, where $k_1 + k_2 = k$ and there is exactly one character in every row (see Figure 2).

$$\begin{array}{cc}
a_1 & - \\
a_2 & - \\
\dots & \dots \\
a_{k_1} & - \\
- & a_{k_1+1} \\
- & a_{k_1+2} \\
\dots & \dots \\
- & a_k
\end{array}$$

Figure 2. A multiple alignment for k length-1 sequences on two columns.

It can be assumed without loss of generality that the sequences in the first column are $a_1, a_2, \dots, a_{k_1}$ and every other sequence are placed in the second row. If the cost of the first column of $\mathcal{A}$ is denoted by $cost(l_1)$, then

$$cost(l_1) = \sum_{i=1}^{k_1} \sum_{\substack{j=2 \\ i<j}}^{k_1} d(a_i, a_j) + k_2 \sum_{i=1}^{k_1} d(a_i, -).$$

Similiarly, the cost of the second column is

$$cost(l_2) = \sum_{i=k_1+1}^{k} \sum_{\substack{j=k_1+2 \\ i<j}}^{k} d(a_i, a_j) + k_1 \sum_{j=k_1+1}^{k} d(a_j, -),$$

and $cost(\mathcal{A}) = cost(l_1) + cost(l_2)$.

A lower bound for $cost(\mathcal{A})$ can be determined by pairing the $d(a_i, -)$ summands in $cost(l_1)$ to the summands of same form in $cost(l_2)$ and using triangle inequality. E.g., for a fix i ($1 \leq i \leq k_1$) and $\forall j : k_1+1 \leq j \leq k$, it is true that $d(a_i, -)+d(a_j, -) \geq d(a_i, a_j)$, so $k_2 d(a_i, -) + \sum_{j=k_1+1}^{k} d(a_j, -) \geq \sum_{j=k_1+1}^{k} d(a_i, a_j)$. It is useful to notice that the summands on the right side of this inequality are exactly those ones that are not included in $cost(c_1)$ when we consider summands of the form of $d(a_i, a_j)$ for this fix i.

By considering this inequality for every $1 \leq i \leq k_1$, the following lower bound can be given:

$$k_2 \sum_{i=1}^{k_1} d(a_i, -) + k_1 \sum_{j=k_1+1}^{k} d(a_j, -) \geq \sum_{i=1}^{k_1} \sum_{j=k_1+1}^{k} d(a_i, a_j)$$

This is implying that

$$cost(\mathcal{A}) \geq \sum_{i=1}^{k_1} \sum_{\substack{j=2 \\ i<j}}^{k_1} d(a_i, a_j) + \sum_{i=k_1+1}^{k} \sum_{\substack{j=k_1+2 \\ i<j}}^{k} d(a_i, a_j) + \sum_{i=1}^{k_1} \sum_{j=k_1+1}^{k} d(a_i, a_j) =$$

$$= \sum_{i=1}^{k} \sum_{\substack{j=2 \\ i<j}}^{k} d(a_i, a_j) = C.$$

It is assumed that the trivial alignment with cost C is not optimal, therefore $\mathcal{A}$ can not be an optimal alignment of S. By this contradiction, it is proved that an optimal alignment of S can not have exactly 2 columns.

Using induction, let be assumed that it is shown $\forall i : 2 \leq i < l$ that an optimal alignment can not have exactly i columns, and let $\mathcal{A}$ be an optimal alignment with l columns. Considering the cost of the first two columns of $\mathcal{A}$, there are k_1 sequences in the first column and k_2 sequences in the second one. It is enough to prove that by merging these two columns, the cost of the new alignment is lower than the cost of $\mathcal{A}$. The cost of these columns in $\mathcal{A}$ (as it can be seen in Figure 3) is equal to

$$\sum_{i=1}^{k_1} \sum_{\substack{j=2 \\ i<j}}^{k_1} d(a_i, a_j) + (k - k_1) \sum_{i=1}^{k_1} d(a_i, -) + \sum_{i=k_1+1}^{k_2} \sum_{\substack{j=k_1+1 \\ i<j}}^{k_2} d(a_i, a_j) + (k - k_2) \sum_{i=k_1+1}^{k_2} d(a_i, -).$$

$$\begin{array}{cc}
a_1 & - \\
a_2 & - \\
\dots & \dots \\
a_{k_1} & - \\
- & a_{k_1+1} \\
- & a_{k_1+2} \\
\dots & \dots \\
- & a_{k_1+k_2} \\
- & - \\
\dots & \dots \\
- & -
\end{array}$$

Figure 3. The first two columns of $\mathcal{A}$.

Let us focus on the first $k' = k_1 + k_2$ characters of these columns. It is an alignment of $\{a_1, a_2, \dots, a_{k'}\}$ on two columns and it was shown that if these sequences are aligned trivially instead of using two columns, then the cost of the alignment can not be higher. It means the following:

$$\sum_{i=1}^{k_1} \sum_{\substack{j=2 \\ i<j}}^{k_1} d(a_i, a_j) + k_2 \sum_{i=1}^{k_1} d(a_i, -) + \sum_{i=k_1+1}^{k'} \sum_{\substack{j=k_1+1 \\ i<j}}^{k'} d(a_i, a_j) +$$

$$+ k_1 \sum_{i=k_1+1}^{k'} d(a_i, -) + (k - k') \sum_{i=1}^{k_1} d(a_i, -) + (k - k') \sum_{i=k_1+1}^{k'} d(a_i, -) \geq$$

$$\geq \sum_{i=1}^{k'} \sum_{\substack{j=2 \\ i<j}}^{k'} d(a_i, a_j) + (k - k') \sum_{i=1}^{k'} d(a_i, -).$$

On the left side of this inequality, there is the cost of the first two columns of $\mathcal{A}$, while on the right side, there is the cost of the column that is constructed by merging the first two columns of $\mathcal{A}$. Therefore, a lower bound for $cost(\mathcal{A})$ is given by an alignment that has $l - 1$ columns, implying that $\mathcal{A}$ can not be optimal. $\square$

3.2. Multiple sequence alignment for length-2 sequences

In this section, it will be shown that using unit metric, a set of length-2 sequences can not be aligned with less cost than their trivial alignment, however, this statement does not hold using arbitrary metric.

Theorem 4 *Using unit metric, no multiple sequence alignment for length-2 sequences exists that has less cost than their trivial alignment. Suppose we align k different sequences ($s_1 = a_{i_1}a_{i_{k+1}}, s_2 = a_{i_2}a_{i_{k+2}}, \dots, s_k = a_{i_k}a_{i_{2k}}$), then the cost of an optimal alignment is* $\displaystyle\sum_{j=1}^{k}\sum_{\substack{l=2\\ j<l}}^{k} d(a_{i_j}, a_{i_l}) + \sum_{j=k+1}^{2k}\sum_{\substack{l=k+2\\ j<l}}^{2k} d(a_{i_j}, a_{i_l})$.

Proof. Let S denote the set of sequences that need to be aligned. It is clear that the trivial alignment of S has the cost written above, so this lower bound is accessible. In other words, it is enough to prove that for any S, a non-trivial alignment can not have less cost than the trivial one.

Let $\mathcal{A}$ be an alignment of S on t columns where $t \geq 3$. Let the rows of $\mathcal{A}$ be permuted so that those aligned sequences, where the indices of the two non-gap characters are the same, are placed under each other, forming a block of sequences (by this operation, the cost of $\mathcal{A}$ does not change). In every row of $\mathcal{A}$, there are exactly two characters and $t-2$ gaps, so there can be $\binom{t}{2}$ types of aligned sequences in $\mathcal{A}$, considering only the positions of the characters in a row. This implies that there will be $\binom{t}{2}$ (not necessarily non-empty) blocks after permuting the rows of $\mathcal{A}$. (E.g., if $t = 4$, then there are $\binom{4}{2} = 6$ blocks after row permuting, see Figure 4.)

After making this block setting, it is clear that there are six types of aligned character pairs in $\mathcal{A}$:

i) first characters of some sequences aligned with other sequences' first characters;

ii) first characters of some sequences aligned with other sequences' second characters;

iii) first characters of some sequences aligned with gaps;

iv) second characters of some sequences aligned with other sequences' second characters;

v) second characters of some sequences aligned with gaps;

vi) gaps aligned with gaps.

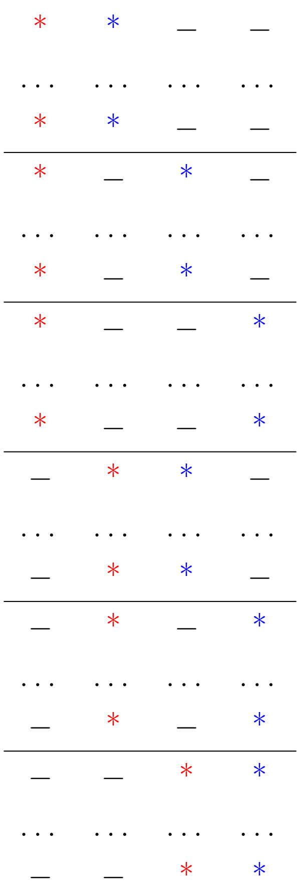

Figure 4. The structure of $\mathcal{A}$ after permuting its rows and making its block setting if $t = 4$. The red stars denote the sequences' first characters, while the blue ones denote their second letters. During the proof, an upper bound is given for the cost of aligning letters with the same color that are not aligned in $\mathcal{A}$ by using character-gap alignment costs that are included in cost($\mathcal{A}$).

In the trivial alignment $\mathcal{T}$, there are only pairs of type i) and iv), moreover, *every* sequence's first character is aligned with each other in $\mathcal{T}$ (and it holds similiarly for *every* second character of the sequences of S). Nevertheless, in a non-trivial alignment $\mathcal{A}$, there are aligned sequences whose first or second characters are not aligned with each other in $\mathcal{A}$. This implies that it is enough to give an upper bound for the cost of these characters in $\mathcal{T}$ that are aligned with each other in $\mathcal{T}$ but are not aligned with each other in $\mathcal{A}$, using parts of cost($\mathcal{A}$) for this bound (see Figure 5). (Because every part of cost($\mathcal{A}$) is nonnegative, if a bijection can be given between the letter-letter alignments in $\mathcal{T}$ that are not aligned in $\mathcal{A}$ and some other alignments of characters of $\mathcal{A}$ (not excluded character-gap alignments) so that the latter alignments have always at least as much cost as the former ones, then it means that $\mathrm{cost}(\mathcal{A}) \geq \mathrm{cost}(\mathcal{T})$.)

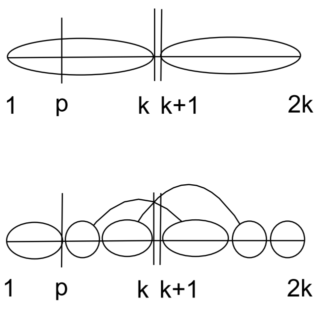

Figure 5. The general structure of character-character alignments of $\mathcal{T}$ (above) and $\mathcal{A}$, if $t = 4$. An ellipse is symbolizing that every pair of characters that are contained by the same ellipse are aligned with each other in the given alignment, moreover, in the figure of $\mathcal{A}$, two edge-connected ellipses are also aligned with each other. It can be seen that in $\mathcal{T}$, every pair of the first characters of sequences are aligned with each other and this statement is true for every pair of second characters of sequences, too. However, in $\mathcal{A}$, any pairs of the form (a_{i_j}, a_{i_l}) will not be aligned, where $1 \leq i_j \leq p$ and $p+1 \leq i_l \leq k$, implying that the cost of $d(a_{i_j}, a_{i_l})$, which is a part of $\mathrm{cost}(\mathcal{T})$ but not a part of $\mathrm{cost}(\mathcal{A})$, must be overestimated with a part of $\mathrm{cost}(\mathcal{A})$.

If d denotes the unit metric, then the following inequality holds for every pair of sets P, R on arbitrary alphabet (where P and R can contain a letter more than once):

$$\sum_{a_{i_j} \in P} \sum_{a_{i_l} \in R} d(a_{i_j}, a_{i_l}) \leq |P| \sum_{a_{i_l} \in R} d(a_{i_l}, -) = |P||R|.$$

Using this inequality, a bijection mentioned above can be given: first, let be considered two sequences whose first characters (a_i and a_j) are not aligned in $\mathcal{A}$. (It can be assumed that a_j has bigger column index.) This implies that the element that is in the intersection of the row of a_j and the column of a_i must be a gap. $d(a_i, a_j) \leq d(a_i, -)$, so the cost of the alignment of a_i and a_j in $\mathcal{T}$ can be estimated by the cost of the alignment of two characters in $\mathcal{A}$.

Similiarly, if two sequences are considered whose second characters (a_i and a_j) are not aligned in $\mathcal{A}$, then (assuming that a_j has bigger column index) the element in the intersection of the row of a_i and the column of a_j must be a gap. The same estimation can be given like before, meaning that the cost of the alignment of a_i and a_j in $\mathcal{T}$ is less or equal to the cost of a character-gap alignment in $\mathcal{A}$.

Considering the block setting of $\mathcal{A}$, let B_i and B_j two blocks whose sequences' first characters are not aligned in $\mathcal{A}$. Assuming that the first characters of sequences in B_j have bigger column index, there must be $|B_j|$ gaps in the intersection of the column of the first characters of sequences in B_i and the rows of B_j. If we denote the first letters of the sequences of B_i (B_j) by a_{b_i} (a_{b_j}), then (because of the statements of the latter two paragraphs) the following holds:

$$\sum_{b_i \in B_i} \sum_{b_j \in B_j} d(a_{b_i}, a_{b_j}) \leq |B_j| \sum_{b_i \in B_i} d(a_{b_i}, -) = |B_i||B_j|$$

1	2	–	–
1	–	2	–
1	–	–	2
–	1	2	–
–	1	–	2
–	–	1	2

Figure 6. The block setting of $\mathcal{A}$ if $t = 4$, denoting only that an element is the first/second character of its aligned sequence or a gap. E.g., the first element of the first row in the block setting and the second element of the fourth row (which are denoting the first characters of some sequences) are not aligned in $\mathcal{A}$, so the cost of their alignment with each other, which is a part of cost($\mathcal{T}$) but not a part of cost($\mathcal{A}$), must be overestimated with a part of cost($\mathcal{A}$), namely, with the cost of aligning the block setting's first element of the first row with the gaps in the first element of the fourth row.

Besides that, a similiar result can be established if we consider two blocks whose sequences' second characters are not aligned, using the gaps of the block that has the column with smaller column index (see Figure 6). By these estimations, it is clear that this assignment between the character-character alignments in $\mathcal{T}$ that are not present in $\mathcal{A}$ and character-gap alignments in $\mathcal{A}$ implies that the latter costs in $\mathcal{A}$ can not be less than the corresponding costs in $\mathcal{T}$. It also must be examined that this assignment is a bijection, i. e. there are no character-gap alignments that are used multiple times.

A set of gaps in the block setting are considered in an estimation if and only if some characters in the block that is containing these gaps and some characters from another block that are aligned in the same column must be aligned in $\mathcal{T}$ but they are not aligned in $\mathcal{A}$. This is implying that these gaps are not used in estimations like above more times than the alignment of this gap set with the rest of the given column, therefore the former assignment is a bijection, implying that $\text{cost}(\mathcal{A}) \geq \text{cost}(\mathcal{T})$. $\square$

Note. It is worthy of note that during the proof, the following special property of unit metric has been used only: $\forall a_i, a_j \in \Sigma : d(a_i, a_j) \leq d(a_i, -)$. It follows that no alignment for a set of length-2 sequences may exist with less cost than their trivial alignment, if a metric that has the same property is being used.

As the next example shows, trivial alignment will not always be optimal for length-2 sequences if an arbitrary metric can be used. Let Σ contain two characters (C and G) with the same metric on Σ as in the Example *ii)* at the end of Section 2, moreover, let S be also the same: $S = \{CG, GC, GG\}$. The trivial alignment of S has a cost of 8, but as it has been shown, there is an alignment of S that has only cost of 6 (see Figure 7).

```
C  G      –  C  G
G  C      G  C  –
G  G      G  –  G
```

Figure 7. The trivial and an optimal alignment of S.

Note. In Section 2, it was shown that we can easily determine the minimum cost of a set to be aligned if it includes only length-1 sequences, moreover, we also can construct an optimal alignment in the most trivial way using any metric. We have also seen that for length-2 sequences, the trivial alignment is optimal if unit metric is used but it is not optimal for arbitrary metric. Besides that, it is also known that trivial alignment is not always optimal for length-3 sequences even using unit metric.

As in Example i) at the end of Section 2, let S be the follow: $S = \{CCG, GCG, CGC\}$. Using unit metric, the cost of the trivial alignment is 6, but it is not optimal: as we have seen, there is a non-trivial alignment $\mathcal{A}$ of S so that cost($\mathcal{A}$) is only 5 (see Figure 8).

$$\begin{array}{ccc} \mathrm{C} & \mathrm{C} & \mathrm{G} \\ \mathrm{G} & \mathrm{C} & \mathrm{G} \\ \mathrm{C} & \mathrm{G} & \mathrm{C} \end{array} \qquad \begin{array}{cccc} \mathrm{C} & \mathrm{C} & \mathrm{G} & - \\ \mathrm{G} & \mathrm{C} & \mathrm{G} & - \\ - & \mathrm{C} & \mathrm{G} & \mathrm{C} \end{array}$$

Figure 8. The trivial and an optimal alignment of S.

4. General results in sequence alignment problems for arbitrary length sequences

In this chapter, some propositions about the structure of an optimal alignment for a given set of sequences are examined, in order to construct a new heuristic algorithm in some special cases of this problem (see Chapter 5). Some results with regard to these propositions will be shown: some of them could be proven, but in many cases (even for some of the most plausible ones) counterexamples could be given.

Note. If one is looking for some structure in the optimal alignments of a given set of sequences, one might think that if we permute the columns of an optimal alignment, then the new alignment will be optimal for the new permuted sequences. This conjecture is not true, as the following counterexample can justify this statement.

$$\begin{array}{cccc} \mathrm{A} & \mathrm{B} & \mathrm{A} & - \\ - & \mathrm{B} & \mathrm{A} & \mathrm{B} \end{array} \qquad \begin{array}{cccc} \mathrm{A} & \mathrm{A} & \mathrm{B} & - \\ - & \mathrm{A} & \mathrm{B} & \mathrm{B} \end{array} \qquad \begin{array}{ccc} \mathrm{A} & \mathrm{A} & \mathrm{B} \\ \mathrm{A} & \mathrm{B} & \mathrm{B} \end{array}$$

Figure 9. From left to right: an optimal alignment $\mathcal{A}$ of S; the new alignment of S' after permuting the columns of $\mathcal{A}$; an optimal alignment of S'.

Using unit metric, an optimal alignment with cost 2 for the set $S = \{ABA, BAB\}$ can be seen on the left side of Figure 9. If the second and third column of this alignment are transposed, we get an alignment with the same cost 2 for the set $S' = \{AAB, ABB\}$. This alignment is not optimal: on the right figure, there is an alignment for S' with cost 1.

Note. Looking for some structure in the MSA problem and considering only sequences of equal length, it can be a plausible idea to permute the columns of the sequences written under each other without gaps in a predetermined (e.g., lexicographical) order. If the value of an optimal alignment for the permuted sequences would not be changed by this operation, then it would be enough to examine the MSA problem only for these special sequences of a "canonical" form.

The conjecture above is not true, since a counterexample can be given. Let S be the same as in the previous note: $S = \{ABA, BAB\}$. As we have seen, S has an optimal alignment with cost 2, using unit metric. If the first two columns are transposed in the trivial alignment of S, then the new set S'' of sequences will be $S'' = \{BAA, ABB\}$. It can be shown that any optimal alignment for S'' has a cost of 3. (see Figure 10)

$$\begin{array}{cccc} \mathrm{A} & \mathrm{B} & \mathrm{A} & - \\ - & \mathrm{B} & \mathrm{A} & \mathrm{B} \end{array} \qquad \begin{array}{ccc} \mathrm{B} & \mathrm{A} & \mathrm{A} \\ \mathrm{A} & \mathrm{B} & \mathrm{B} \end{array}$$

Figure 10. An optimal alignment of S with cost 2 and an optimal alignment of S'' with cost 3.

Note. One might think that if a set of sequences with equal length have the same subsequences on the same positions, then these subsequences must be aligned the same way in an optimal alignment. This heuristic idea is also false, it can be seen from the next counterexample.

Let S be the next set of sequences: $S = \{AAAAABBBAACCC, BBBAACCCDDDDD, CCCAABBBAACCC\}$. In every sequence in S, the fourth and fifth character is "AA". If only those alignments are considered where these subsequences are aligned under each other, then we have to align the prefixes and suffixes optimally. This yields the trivial alignment of the sequences with cost 25, but this is not an optimal alignment of S: there can be shown an alignment for this set with cost 23 (see Figure 11).

A	A	A	A	A	B	B	B	A	A	C	C	C
B	B	B	A	A	C	C	C	D	D	D	D	D
C	C	C	A	A	B	B	B	A	A	C	C	C

A	A	A	A	A	B	B	B	A	A	C	C	C	–	–	–	–	–
–	–	–	–	–	B	B	B	A	A	C	C	C	D	D	D	D	D
C	C	C	A	A	B	B	B	A	A	C	C	C	–	–	–	–	–

Figure 11. The trivial alignment of S with cost 25 which is optimal if we consider only alignments where the identical subsequences in the fourth an fifth position are under each other; and an optimal alignment of S with cost 23.

The counterexamples above show that many plausible ideas are not true for the MSA problem. In the next theorem, one of these propositions is shown to be true in a special case.

Theorem 5 *Using unit metric, if we align two sequences with identical prefixes, then there exists an optimal alignment in which these prefixes are aligned under each other.*

Proof. It is enough to show that if the first character of the sequences are identical but their second character is different, then there is an optimal alignment where the first characters are aligned under each other.

Let S be the following two sequences: $s_1 = ABs_1(3)\dots s_1(n)$, $s_2 = ACs_2(3)\dots s_2(m)$ and denote $s'_1 = Bs_1(3)\dots s_1(n), s'_2 = Cs_2(3)\dots s_2(m)$. We can construct every optimal alignment of $S' = \{s'_1, s'_2\}$ by Needleman-Wunsch algorithm. Let T' denote the matrix that is created by removing the first row and the first column from the Needleman-Wunsch table A' of S'. With this notation, the structure of this table can be seen on Figure 12. Every optimal alignment of S' can be corresponded to a path in this table that starts from the lower right corner of A' and ends on the 0 cell in the upper left corner.

To prove the proposition, it is enough to show that for the second row and the second column of the Needleman-Wunsch table A of S the following holds: $A[2, j] = j - 2$ ($j = 2, \dots, n = |s_1|$) and $A[j, 2] = j - 2$ ($j = 2, \dots, m = |s_2|$). To see this, we can note that

	$*$	B	$s_1(3)$	$\dots$	$s_1(n)$
$*$	0	1	2	$\dots$	$n-1$
C	1				
$s_2(3)$	2				
$\dots$	$\dots$			T'	
$s_2(m)$	$m-1$				

Figure 12. Structure of the Needleman-Wunsch table of S'.

$A[2,2] = 0$ because $A[1,1] = 0$ and $A[1,2] = A[2,1] = 1$ by definition and there is a match in the first positions of sequences. Using induction, the dynamic programming rule yields: $A[2,j]$ =min$\{A[1,j-1](+1), A[1,j]+1, A[2,j-1]+1\}$ =min$\{j-2(+1), j, j-2\} = j-2$. (The $(+1)$ denotes that if there is a mismatch for these characters, then this value must be increased by 1, otherwise not.)

Similiarly, $A[j,2] = j-2$ also holds for the elements of the second column, therefore the structure of A can be seen on Figure 13.

	$*$	A	B	$s_1(3)$	$\dots$	$s_1(n)$
$*$	0	1	2	3	$\dots$	n
A	1	0	1	2	$\dots$	$n-1$
C	2	1				
$s_2(3)$	3	2				
$\dots$	$\dots$	$\dots$			T'	
$s_2(m)$	m	$m-1$				

Figure 13. Structure of the Needleman-Wunsch table of S.

It is clear that if we remove the first row and the first column of A, then A' is remained. As a part of A, A' contains every pointer that it had in the separate alignment for S', hence there exists an optimal alignment of S in A that has $A[2,2]$ as its penultimate cell. This alignment ends with a match on the cell $A[1,1]$, therefore the first characters are aligned under each other in this alignment.

Note. It is worth to note that only the following properties of the unit metric were used: $\forall\ a_i \in \Sigma\ d(a_i, -) = c$ for some constant c and $d(a_i, a_j) = 0 \Leftrightarrow a_i = a_j$. It implies that for every cost function that has these properties, Theorem 5 holds.

5. A new heuristic algorithm for the multiple sequence alignment problem

In this section, a new heuristic algorithm is presented, regarding the multiple sequence alignment problem. This heuristic is based on the dynamic programming algorithm solution for the pairwise alignment problem: the aligned sequences from the output of Needleman-Wunsch algorithm are directly used in the construction of the multiple alignment for the given sequences.

In the first subsection, a basic version of this heuristic are described, and after that, two modified versions of this basic algorithm are also featured. After the exposition of these three heuristics, their results for some test sequence sets are compared with the results given by Clustal, one of the most widely used MSA software.

5.1. A basic heuristic version

Let S be the set of n input sequences: $S = \{s_1, s_2, \ldots, s_k\}$. The general method of this heuristic for the MSA problem can be partitioned to the following four steps:

(1) Needleman-Wunsch algorithm is performed on every pair of sequences from S and $\forall\ 1 \leq i \leq k : k-1$ output aligned sequences are stored for s_i in the list S_i^*.

(2) Let s_i^* denote the most common element of S_i^* (if this element is not unique, then the algorithm works over with a random chosen element among the most common ones). The set S^* consisting of s_i^*'s ($\forall 1 \leq i \leq k$) is used for the initiate step of constructing a heuristic multiple alignment $\mathcal{A}$.

(3) Let l_{max} denote the length of a maximum length element of S^*. $\forall\ 1 \leq i \leq k$: if the length of s_i^* is less than l_{max}, then s_i^* is expanded with inserted gaps at its end until its length is equal with l_{max}. Let s_i' denote this sequence expanded from s_i^*.

(4) The multiple alignment of S given by this heuristic: $\mathcal{A} = \{s_1', s_2', \ldots, s_k'\}$.

Example. Let S be the the following set: $S = \{BBA, BABA, ABBAB\}$. The Needleman-Wunsch algorithm for BBA and $BABA$ has these aligned output sequences as an optimal alignment: $\{B-BA, BABA\}$, so $B-BA$ will be an element in S_1^* and $BABA \in S_2^*$. After every optimal pairwise alignment is computed, the following sets are resulted: $S_1^* = \{B-BA, -BBA-\}, S_2^* = \{BABA, --BABA\}, S_3^* = \{ABBAB, ABBAB-\}$. The frequencies are equal in every list, meaning that the a random element from every S_i^* is chosen to S^*, therefore $S^* = \{B-BA, BABA, ABBAB\}$. $l_{max} = 5$, but the length of s_1^* and s_2^* is smaller than l_{max}, hence inserting gaps is required for these sequences. From the modified aligned sequences, the multiple aligment given by this heuristic will be the following: $\mathcal{A} = \{B-BA-, BABA-, ABBAB\}$.

Note. As it can be established from the algorithm description (and it is supported by the results of the practical testing of this heuristic), an alignment close to optimal can be computed by this approach, if the multiple alignment that can be mechanically combined from the optimal pairwise dynamic programming solutions is "close" to an optimal MSA, and it follows that the sum of the pairwise optimal costs for the sequences is relatively close to the cost of an optimal alignment.

For example, this heuristic could not find an optimal alignment with cost 5 for the set $\{AAB, BAB, ABA\}$ because $S_3^* = \{ABA, ABA-\}$, and in an optimal alignment, $s_3' = -ABA$ should be required. s_3' can not be prepared from any element of S_3^* by this algorithm, so the output MSA is not optimal (in fact, its cost 8 is even worse than the cost of the trivial alignment which is equal to 6).

Considering another example, if the input sequence set is $S = \{BABA, \quad AAAA, BBBA\}$, then the output of the Needleman-Wunsch algorithm for every pair is the pair of sequences without gaps, meaning that $S^* = S$ and from this, $\mathcal{A} = S$. In this case, an optimal alignment for S is easily computed from the input sequences, since it is corresponded with the trivial alignment, and the heuristic also could find it because it was close enough to the multiple alignment combined from the pairwise optimums.

5.2. Two modified versions of the basic heuristic

Based on the basic version presented in the previous subsection, two modified heuristic versions has also been processed.

In the first modification, before the first step of the original algorithm, input sequences are sorted in descending order by their length as a preprocessing step. By this supplement in the heuristic, an unfavourable attribute of basic algorithm could be eliminated: the aligned sequences of the output alignment will not depend on the order of input sequences, in spite of that output of basic heuristic can be depended by the order of input set. Therefore, this ordered heuristic can yield a unique solution for any input sequence set.

In the second modified heuristic version, a randomization approach has been applied. In the third step of the original heuristic, it was strictly assigned that the inserted gaps could be located only at the end of a given aligned sequence. In this modification, a gap to be inserted can be located with probability $\frac{1}{2} - \frac{1}{2}$ at either end of the actual sequence. By this change in the algorithm, M is not previously determined by the elements of S^*: many different multiple alignments can be derived from the same S^* by the randomized heuristic.

5.3. Results of the basic heuristic and its two modified versions

In this section, the results of this basic heuristic on some randomly generated test sequence sets ($k = 3$) using unit metric are presented, compared with the alignments provided by the Clustal software. For Clustal results, I have used a webserver of Bielefeld University [9] with the following parameters:

- type of alignment: slow;
- DNA weight matrix: IUB;
- gap open = 1;
- gap extension = 0.001;
- gap distance = 1;
- end gap penalty: disabled;

- type of iteration: alignment;
- number of iterations: 10;
- clustering type: neighbour joining.

IUB matrix is corresponding practically to unit metric. "Gap open", "gap extension" and "end gap penalty" denote extra costs for opening a sequence of gaps, extending one or closing one, respectively. As these kind of costs are not considered by this heuristic, their values has been adjusted to available minimum. "Gap distance" denotes the cost of a letter and a gap. By setting type of iteration and clustering as above, classic Clustal method for sequence alignment with neighbour joining and guide trees has been applied. Test sequences contained only two kinds of letters (A and B) and their length was restricted for the integer values with a minimum 4 and a maximum 7.

For 50 test sequence sets, results of these two algorithms have been compared (detailed results are contained by Table 1). In these cases, the cost provided by basic heuristic was less with $\frac{2}{25}$ on the average than the cost from Clustal. In the 68 % of all test cases, the alignment of basic heuristic was at least as good as the Clustal one. Moreover, in the 40 % of tests, output of basic heuristic had strictly lower cost than Clustal's.

The two modified versions of the basic heuristic have also tested for the same sequence sets as the basic version. The sorted heuristic (see Table 2) had the same average efficiency compared to Clustal as the basic heuristic with $\frac{2}{25}$ lower average costs. For five test sets, output of sorted heuristic had a lower cost than basic heuristic, but in three cases, the original algorithm had a better MSA output.

The randomized algorithm version had a worse average result than basic heuristic for these 50 test cases, moreover, its outputs had a $\frac{3}{25}$ higher average cost than Clustal's. Compared to basic heuristic, the algorithm could find a better alignment for six test sets by randomization, but its efficiency in these cases could not neutralize that for eight sets, its output had a higher cost than the original algorithm (see Table 3).

Sequences	Clustal result	Basic heuristic result	Lower cost	Difference
$BABBBAA, ABBBAAA, ABABAB$	8	15	C	+7
$ABBB, ABABABB, ABAAB$	8	13	C	+5
$BBBBA, BBBAAAA, ABBA$	10	14	C	+4
$AAAABA, ABABBBB, AAABB$	9	12	C	+3
$BABBA, BABB, BAAAAB$	8	11	C	+3
$ABAAAB, ABABB, ABBAAAB$	6	9	C	+3
$ABAABBB, AABA, ABBAA$	12	14	C	+2
$BABBBAA, ABBAAAB, BAABBB$	12	14	C	+2
$BBABA, BAABBA, ABBAAB$	11	13	C	+2
$AABBABB, BABAAB, ABBBB$	9	11	C	+2
$BAAA, BBAABAB, ABBABAA$	10	11	C	+1
$BABAAA, ABBBA, BABB$	9	10	C	+1
$AAAABB, BAAAABB, AAABBBA$	8	9	C	+1
$BABBBAB, BBABAB, BABB$	8	9	C	+1
$AABBB, BBAAB, BBAB$	8	9	C	+1
$ABBBBA, BABBBA, BBBAB$	8	9	C	+1
$AABBAAB, BABA, BBBBBB$	13	13	T	0
$ABBBABB, AAABBBB, BBBAA$	12	12	T	0
$BAABAAB, ABABBAA, ABABBB$	10	10	T	0
$AABBA, BBABA, ABABBBA$	10	10	T	0
$BBBBABA, BABA, BBAAB$	10	10	T	0
$BAABAAB, ABABBAA, ABABBB$	10	10	T	0
$ABBABAA, BBAAA, AABA$	10	10	T	0
$BAAAA, AABBAB, AABBB$	10	10	T	0
$AABABAA, BBABA, BBBABB$	9	9	T	0
$BAABAA, BBAA, ABABA$	8	8	T	0
$BAAA, AAAABAA, AAAA$	8	8	T	0
$BABBAB, BBBBBAB, BBABBA$	6	6	T	0
$BBAAB, BBBAAB, BABAAA$	6	6	T	0
$BBBABA, BBABA, ABBAA$	5	5	T	0
$AABA, BBBBBAA, BBABBB$	14	13	H	-1
$AABAA, BABABBB, BAAA$	13	12	H	-1
$BBBAABB, BABBBA, ABAAAA$	13	12	H	-1
$BBABB, ABAAAA, AABB$	13	12	H	-1
$BABA, AAABAA, ABBBB$	12	11	H	-1
$AABABBA, BAAABB, BAABAA$	10	9	H	-1
$BBBB, BBAA, ABBBB$	10	9	H	-1
$BAABA, AABBBBA, AABBABA$	9	8	H	-1
$BAAABAA, BBBAA, BBAA$	8	7	H	-1
$AAABBA, BBABBBB, BBAA$	15	13	H	-2
$BBABBA, AABAB, BAABBBA$	12	10	H	-2
$BBABBBA, ABBBA, AABBBBB$	12	10	H	-2
$BABA, AAAA, BBBA$	8	6	H	-2
$BBAAA, BBAABAB, AAABAA$	14	11	H	-3
$AABBBBB, BABBAAB, AAAABB$	14	11	H	-3
$ABBAAA, BAABB, BBBBB$	13	10	H	-3
$BBBAAAB, ABBBBAB, ABBABA$	11	8	H	-3
$AABA, BABBBBB, ABAAAA$	17	13	H	-4
$BABA, BBABBB, BBBBAAB$	14	10	H	-4
$BBBBBB, BAAA, BBBBB$	16	10	H	-6

Table 1. Comparison between the results of Clustal and basic heuristic for 50 test sequence sets. In the columns, from left to the right: test set; alignment cost provided by Clustal; alignment cost provided by basic heuristic; abbreviation of the algorithm providing lower cost (T: the costs are equal); difference between costs of alignment from heuristic and from Clustal.

Sequences	Clustal results	Basic heuristic result	Difference between heuristic and Clustal	Sorted heuristic result	Difference between sorted heuristic and Clustal	Lower cost
$BABBBAA, ABBBAAA, ABABAB$	8	15	+7	15	+7	C
$ABBB, ABABABB, ABAAB$	8	13	+5	10	+2	C
$BBBBA, BBBAAAA, ABBA$	10	14	+4	14	+4	C
$AAAABA, ABABBBB, AAABB$	9	12	+3	9	0	T
$BABBA, BABB, BAAAAB$	8	11	+3	11	+3	C
$ABAAAB, ABABB, ABBAAAB$	6	9	+3	6	0	T
$ABAABBB, AABA, ABBAA$	12	14	+2	14	+2	C
$BABBBAA, ABBAAAB, BAABBB$	12	14	+2	14	+2	C
$BBABA, BAABBA, ABBAAB$	11	13	+2	13	+2	C
$AABBABB, BABAAB, ABBBB$	9	11	+2	11	+2	C
$BAAA, BBAABAB, ABBABAA$	10	11	+1	11	+1	C
$BABAAA, ABBBA, BABB$	9	10	+1	10	+1	C
$AAAABB, BAAAABB, AAABBBA$	8	9	+1	9	+1	C
$BABBBAB, BBABAB, BABB$	8	9	+1	9	+1	C
$AABBB, BBAAB, BBAB$	8	9	+1	9	+1	C
$ABBBBA, BABBBA, BBBAB$	8	9	+1	9	+1	C
$AABBAAB, BABA, BBBBBB$	13	13	0	13	0	T
$ABBBABB, AAABBBB, BBBAA$	12	12	0	12	0	T
$BAABAAB, ABABBAA, ABABBB$	10	10	0	10	0	T
$AABBA, BBABA, ABABBBA$	10	10	0	13	+3	C
$BBBBABA, BABA, BBAAB$	10	10	0	10	0	T
$BAABAAB, ABABBAA, ABABBB$	10	10	0	10	0	T
$ABBABAA, BBAAA, AABA$	10	10	0	10	0	T
$BAAAA, AABBAB, AABBB$	10	10	0	10	0	T
$AABABAA, BBABA, BBBABB$	9	9	0	9	0	T
$BAABAA, BBAA, ABABA$	8	8	0	8	0	T
$BAAA, AAAABAA, AAAA$	8	8	0	11	+3	C
$BABBAB, BBBBBAB, BBABBA$	6	6	0	6	0	T
$BBAAB, BBBAAB, BABAAA$	6	6	0	6	0	T
$BBBABA, BBABA, ABBAA$	5	5	0	5	0	T
$AABA, BBBBBAA, BBABBB$	14	13	-1	12	-2	S
$AABAA, BABABBB, BAAA$	13	12	-1	12	-1	S
$BBBAABB, BABBBA, ABAAAA$	13	12	-1	12	-1	S
$BBABB, ABAAAA, AABB$	13	12	-1	11	-2	S
$BABA, AAABAA, ABBBB$	12	11	-1	11	-1	S
$AABABBA, BAAABB, BAABAA$	10	9	-1	9	-1	S
$BBBB, BBAA, ABBBB$	10	9	-1	9	-1	S
$BAABA, AABBBBA, AABBABA$	9	8	-1	8	-1	S
$BAAABAA, BBBAA, BBAA$	8	7	-1	7	-1	S
$AAABBA, BBABBBB, BBAA$	15	13	-2	13	-2	S
$BBABBA, AABAB, BAABBBA$	12	10	-2	10	-2	S
$BBABBBA, ABBBA, AABBBBB$	12	10	-2	10	-2	S
$BABA, AAAA, BBBA$	8	6	-2	6	-2	S
$BBAAA, BBAABAB, AAABAA$	14	11	-3	11	-3	S
$AABBBBB, BABBAAB, AAAABB$	14	11	-3	11	-3	S
$ABBAAA, BAABB, BBBBB$	13	10	-3	10	-3	S
$BBBAAAB, ABBBBAB, ABBABA$	11	8	-3	8	-3	S
$AABA, BABBBBB, ABAAAA$	17	13	-4	18	+1	C
$BABA, BBABBB, BBBBAAB$	14	10	-4	10	-4	S
$BBBBBB, BAAA, BBBBB$	16	10	-6	10	-6	S

Table 2. Comparison between the results of Clustal, basic heuristic and sorted heuristic for 50 test sequence sets. In the columns, from left to the right: test set; alignment cost provided by Clustal; alignment cost provided by basic heuristic; difference between costs of alignment from basic heuristic and from Clustal; alignment cost provided by sorted heuristic; difference between costs of alignment from sorted heuristic and from Clustal; abbreviation of the algorithm providing lower cost (C: Clustal, S: sorted heuristic, T: costs are equal).

Sequences	Clustal results	Basic heuristic result	Difference between heuristic and Clustal	Randomized heuristic result	Difference between randomized heuristic and Clustal	Lower cost
BABBBAA, ABBBAAA, ABABAB	8	15	+7	10	+2	C
ABBB, ABABABB, ABAAB	8	13	+5	13	+5	C
BBBBA, BBBAAAA, ABBA	10	14	+4	14	+4	C
AAAABA, ABABBBB, AAABB	9	12	+3	10	+1	C
BABBA, BABB, BAAAAB	8	11	+3	11	+3	C
ABAAAB, ABABB, ABBAAAB	6	9	+3	12	+6	C
ABAABBB, AABA, ABBAA	12	14	+2	14	+2	C
BABBBAA, ABBAAAB, BAABBB	12	16	+2	14	+4	C
BBABA, BAABBA, ABBAAB	11	13	+2	13	+2	C
AABBABB, BABAAB, ABBBB	9	11	+2	12	+3	C
BAAA, BBAABAB, ABBABAA	10	11	+1	11	+1	C
BABAAA, ABBBA, BABB	9	10	+1	10	+1	C
AAAABB, BAAAABB, AAABBBA	8	9	+1	6	-2	R
BABBBAB, BBABAB, BABB	8	9	+1	9	+1	C
AABBB, BBAAB, BBAB	8	9	+1	9	+1	C
ABBBBA, BABBBA, BBBAB	8	9	+1	9	+1	C
AABBAAB, BABA, BBBBBB	13	13	0	13	0	T
ABBBABB, AAABBBB, BBBAA	12	12	0	12	0	T
BAABAAB, ABABBAA, ABABBB	10	10	0	10	0	T
AABBA, BBABA, ABABBBA	10	10	0	10	0	T
BBBBABA, BABA, BBAAB	10	10	0	10	0	T
BAABAAB, ABABBAA, ABABBB	10	10	0	10	0	T
ABBABAA, BBAAA, AABA	10	10	0	10	0	T
BAAAA, AABBAB, AABBB	10	10	0	10	0	T
AABABAA, BBABA, BBBABB	9	9	0	9	0	T
BAABAA, BBAA, ABABA	8	8	0	13	+5	C
BAAA, AAAABAA, AAAA	8	8	0	10	+2	C
BABBAB, BBBBBAB, BBABBA	6	6	0	6	0	T
BBAAB, BBBAAB, BABAAA	6	6	0	6	0	T
BBBABA, BBABA, ABBAA	5	5	0	5	0	T
AABA, BBBBBAA, BBABBB	14	13	-1	13	-1	R
AABAA, BABABBB, BAAA	13	12	-1	16	+3	C
BBBAABB, BABBBA, ABAAAA	13	12	-1	12	-1	R
BBABB, ABAAAA, AABB	13	12	-1	10	-3	R
BABA, AAABAA, ABBBB	12	11	-1	11	-1	R
AABABBA, BAAABB, BAABAA	10	9	-1	14	+4	C
BBBB, BBAA, ABBBB	10	9	-1	6	-4	R
BAABA, AABBBBA, AABBABA	9	8	-1	8	-1	R
BAAABAA, BBBAA, BBAA	8	7	-1	7	-1	R
AAABBA, BBABBBB, BBAA	15	13	-2	13	-2	R
BBABBA, AABAB, BAABBBA	12	10	-2	12	0	T
BBABBBA, ABBBA, AABBBBB	12	10	-2	10	-2	R
BABA, AAAA, BBBA	8	6	-2	6	-2	R
BBAAA, BBAABAB, AAABAA	14	11	-3	10	-4	R
AABBBBB, BABBAAB, AAAABB	14	11	-3	11	-3	R
ABBAAA, BAABB, BBBBB	13	10	-3	10	-3	R
BBBAAAB, ABBBBAB, ABBABA	11	8	-3	8	-3	R
AABA, BABBBBB, ABAAAA	17	13	-4	13	-4	R
BABA, BBABBB, BBBBAAB	14	10	-4	10	-4	R
BBBBBB, BAAA, BBBBB	16	10	-6	10	-6	R

Table 3. Comparison between the results of Clustal, basic heuristic and randomized insertion heuristic for 50 test sequence sets. In the columns, from left to the right: test set; alignment cost provided by Clustal; alignment cost provided by basic heuristic; difference between costs of alignment from basic heuristic and from Clustal; alignment cost provided by randomized insertion heuristic; difference between costs of alignment from randomized insertion heuristic and from Clustal; abbreviation of the algorithm providing lower cost (C: Clustal, R: randomized insertion heuristic, T: costs are equal).

6. Open questions

In this thesis, it was shown that the multiple sequence alignment problem is "easy" for length-1 sequences and also for length-2 sequences in special cases. Since we know that the general problem is **NP**-complete, it is still an interesting question that for how long sequences MSA starts to become a real hard problem? It is probably another open problem that in case of length-2 sequences, how can those metrics be characterized for which trivial alignment is always optimal for arbitrary alphabet?

Regarding the presented heuristic algorithm, many questions can be raised: e.g., could this approach be this effective for more sequences than three, or for an alphabet containing more letters? It can be another interesting further question that how this heuristic could be improved: as it could be seen in former test results, its efficiency could not be raised by simple randomization, but there could be other useful methods (perhaps some more sophisticated preprocessing steps).

Appendix

Source code for the basic heuristic presented in Chapter 5[1]:

```
from collections import Counter
from random import randint

def Most_Common(lst):
    data = Counter(lst)
    return data.most_common(1)[0][0]

def zeros(shape):
    retval = []
    for x in range(shape[0]):
        retval.append([])
        for y in range(shape[1]):
            retval[-1].append(0)
    return retval

match_award = 0
mismatch_penalty = -1
gap_penalty = -1

def match_score(alpha, beta):
    if alpha == beta:
        return match_award

```

[1]Source code of Needleman-Wunsch algorithm, which is used in this heuristic as a subroutine, was constructed by Aleksandr Levchuk [5].

```
    elif alpha == '-' or beta == '-':
        return gap_penalty
    else:
        return mismatch_penalty


L=['BBBA', 'BABA', 'ABABBB']   #input sequences

Aligned1=[]
Aligned2=[]
Aligned3=[]

for k in range(0,2):
        for l in range(k+1,3):
                seq1=L[k]
                seq2=L[l]
                m, n = len(seq1), len(seq2)
                score = zeros((m+1, n+1))  # DP table

                for i in range(0, m + 1):  # Calculate DP table
                        score[i][0] = gap_penalty * i
                for j in range(0, n + 1):
                        score[0][j] = gap_penalty * j
                for i in range(1, m + 1):
                        for j in range(1, n + 1):
                                match = score[i - 1][j - 1] + match_score(seq1[i-1], seq2[j-1])
                                delete = score[i - 1][j] + gap_penalty
                                insert = score[i][j - 1] + gap_penalty
                                score[i][j] = max(match, delete, insert)



                align1, align2 = '', ''
                i,j = m,n
                while i > 0 and j > 0:
                        score_current = score[i][j]
                        score_diagonal = score[i-1][j-1]
```

```
            score_up = score[i][j-1]
            score_left = score[i-1][j]

            if score_current == score_diagonal + match_score(seq1[i-1], seq2[j-1]):
                align1 += seq1[i-1]
                align2 += seq2[j-1]
                i -= 1
                j -= 1
            elif score_current == score_left + gap_penalty:
                align1 += seq1[i-1]
                align2 += '-'
                i -= 1
            elif score_current == score_up + gap_penalty:
                align1 += '-'
                align2 += seq2[j-1]
                j -= 1


        while i > 0:
            align1 += seq1[i-1]
            align2 += '-'
            i -= 1
        while j > 0:
            align1 += '-'
            align2 += seq2[j-1]
            j -= 1

        align1 = align1[::-1]
        align2 = align2[::-1]

        i,j = 0,0



        symbol = ''
        found = 0
        score = 0

        for i in range(0,len(align1)):
            if align1[i] == align2[i]:
                symbol = symbol + align1[i]
                score += match_score(align1[i], align2[i])
            elif align1[i] != align2[i] and align1[i] != '-' and align2[i] != '-':
                score += match_score(align1[i], align2[i])
                symbol += ' '
                found = 0
            elif align1[i] == '-' or align2[i] == '-':
                symbol += ' '
                score += gap_penalty

        if k==0:
            Aligned1.append(align1)
        else:
            Aligned2.append(align1)
        if l==1:
            Aligned2.append(align2)
        else:
            Aligned3.append(align2)

msa=[]

msa.append(Most_Common(Aligned1))
msa.append(Most_Common(Aligned2))
msa.append(Most_Common(Aligned3))

lengths=[]
for i in range(0,len(msa)):
    lengths.append(len(msa[i]))


maxl=max(lengths)
MSA=[]

```

```
for i in range(0,len(msa)):
        if len(msa[i])==maxl:
                MSA.append(msa[i])
        else:
                c=maxl-len(msa[i])
                gaps=''
                for j in range(0,c):
                        gaps=gaps+'-'
                MSA.append(msa[i]+gaps)
print MSA

MSA_score=0

for k in range(0,2):
        for l in range(k+1,3):
                align1=MSA[k]
                align2=MSA[l]
                m, n = len(seq1), len(seq2)
                symbol = ''
                found = 0
                score = 0
                for i in range(0,len(align1)):
                        if align1[i] == align2[i]:
                                symbol = symbol + align1[i]
                                score += match_score(align1[i], align2[i])
                        elif align1[i] != align2[i] and align1[i] != '-' and align2[i] != '-':
                                score += match_score(align1[i], align2[i])
                                symbol += ' '
                                found = 0
                        elif align1[i] == '-' or align2[i] == '-':
                                symbol += ' '
                                score += gap_penalty
                MSA_score=MSA_score+score

print MSA_score
```

Bibliography

[1] Paola Bonizzoni and Gianluca Della Vedova. The complexity of multiple sequence alignment with SP-score that is a metric. *Theoretical Computer Science*, 259:63–79, 2001.

[2] Isaac Elias. Settling the intractability of multiple alignment. In *Proc. of the 14th Ann. Int. Symp. on Algorithms and Computation (ISAAC)*, pages 352–363. Springer, 2003.

[3] Martin C. Frith, Ulla Hansen, John L. Spouge, and Zhiping Weng. Finding functional sequence elements by multiple local alignment. *Nucleic Acid Research*, 32 (1):189–200, 2004.

[4] Sudhir Kumar and Alan Filipski. Multiple sequence alignment: In pursuit of homologous DNA positions. *Genome Research*, 17:127–135, 2007.

[5] Aleksandr Levchuk. Pairwise alignment in Python. `https://github.com/alevchuk/pairwise-alignment-in-python`, 2015.

[6] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48 (3):443–453, 1970.

[7] Cedric Notredame. Recent evolutions of multiple sequence alignment algorithms. *PLoS Computational Biology*, 3 (8):1405–1408, 2007.

[8] Kristóf Takács. Approximation problems. `http://www.cs.elte.hu/blobs/diplomamunkak/bsc_alkmat/2015/takacs_kristof.pdf`, 2015. Bachelor's thesis, Eötvös Loránd University, Faculty of Science, Department of Computer Science.

[9] Bielefeld University. Clustal webserver. `https://bibiserv.cebitec.uni-bielefeld.de/clustalw`, 2006.

[10] Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1 (4):337–348, 1994.