Technical Report: Measuring fidelity of DTs General Concepts

Paula Muñoz, Javier Troya, Antonio Vallecillo
ITIS Software
University of Málaga (Spain)

Manuel Wimmer
CDL-Mint
Johannes Kepler University (Austria)

Contents

I Introduction

In this document, we present the general concepts related to the trace alignment algorithm presented in the article titled *"Measuring the Fidelity of a Physical and a Digital Twin Using Trace Alignments"*, which is currently under review.

In this report, we introduce all the concepts on which the algorithm is based (alignment, match, mismatch, etc.). Next, we discuss the algorithms that have served as the foundation for our work, namely, the Needleman-Wunsch and BLAST algorithms, along with the concepts from these algorithms that have influenced our approach. Additionally, we provide a detailed introduction to the gap system, the Affine Gap, which is employed in both algorithms. Finally, we provide the formal definitions all the fidelity metrics used to evaluate the results obtained with the algorithm and to reason about the fidelity of the systems.

In order to validate the alignment algorithm, we have considered three Digital Twins Systems developed at different companies and universities. We gathered all the data on the execution of different systems that either serve as DTSs or have the components necessary to build one, i.e., a real system and a virtual one with supposedly equivalent behavior. We consider this latter case in the context of an engineer working with a legacy system who wants to optimize its performance by incorporating a DT. To achieve this, we must first validate that the replica's behavior (simulator, formula, model) is sufficiently faithful to the actual system.

In all the reports associated with the article, we assume that the behavior of the physical system is correct, both in cases where we already have a digital twin (incubator) and in cases where we aim to construct a DT for a legacy system (elevator and robotic arm). Our goal is to ensure that the behavior of the digital replica is as faithful as possible to the physical system within certain margins.

II Definitions

Definition II.1 (Aligment)**.** Given two sequences of snapshots $X = \{x_i\}_{i=1}^{n}$ and $Y = \{y_i\}_{i=1}^{m}$, and a comparison relationship between snapshots "$\approx$", an *alignment* A between X and Y is a set of pairs $A \subseteq \{0..n\} \times \{0..m\}$, which satisfies the following properties (we assume below that i and j will always range between $\{1..n\}$ and $\{1..m\}$, respectively).

- All elements of X are paired with at most one element of Y, i.e., $(i, j_1) \in A \wedge (i, j_2) \in A \Rightarrow j_1 = j_2$.
- All elements of Y are paired with at most one element of X, i.e., $(i_1, j) \in A \wedge (i_2, j) \in A \Rightarrow i_1 = i_2$.
- The set of pairs A must be monotonic, i.e., if $(i_1, j_1) \in A$ and $(i_2, j_2) \in A$ then $i_1 \leq i_2 \Rightarrow j_1 \leq j_2$ and vice-versa: $j_1 \leq j_2 \Rightarrow i_1 \leq i_2$.

The alignment also contains *gaps*, which are the indexes of the elements that have not been paired, i.e.,

- $(i, 0) \in A \Leftrightarrow x_i \in X \wedge \nexists\, j \in \{1..m\} \bullet (i, j) \in A$
- $(0, j) \in A \Leftrightarrow y_j \in Y \wedge \nexists\, i \in \{1..n\} \bullet (i, j) \in A$
- $(0, 0) \notin A$

In the following, G_A will denote the set of gaps of an alignment A, i.e., $G_A = \{(i, 0) \in A\} \cup \{(0, j) \in A\}$.

Note that with this definition, A is well-defined for every (i, j) with $1 \leq i \leq n$ and $1 \leq j \leq m$. Furthermore, every alignment A defines two functions A_X and A_Y with the indexes of the corresponding paired elements

$A_X(i) = j$ if $(i, j) \in A$, or 0 if $(i, 0) \in G_A$

$A_Y(j) = i$ if $(i, j) \in A$, or 0 if $(0, j) \in G_A$

Definition II.2 (Match and mismatch)**.** Given an alignment A, we say that a pair $(i, j) \in A$ is a *match* if $x_i \approx y_j$, and a *mismatch* if $x_i \not\approx y_j$.

Note that more than one alignment can be defined between two sequences. For example, let us suppose that $X = \{a, b, c, d\}$, $Y = \{a, b, d\}$, and $Z = \{a, h, c, d\}$. The following alignments can be defined between them:

$A_1(X, Y) = \{(1, 1), (2, 2), (3, 0), (4, 3)\}$

$A_2(X, Z) = \{(1,1), (2,2), (3,3), (4,4)\}$

$A_3(X, Z) = \{(1,1), (2,0), (0,2), (3,3), (4,4)\}$

The first one contains three matches and one gap, the second one contains three matches and one mismatch $(2, 2)$, and the third one contains three matches and two gaps.

Definition II.3 (Paired and matched subsequences)**.** We will denote by X^A and Y^A the subsequences of X and Y that are *paired* by the alignment with other elements (i.e., not gaps):

$X^A = \{x_i \mid x_i \in X \wedge \exists j \in \{1..m\} \bullet (i, j) \in A\}$

$Y^A = \{y_j \mid y_j \in Y \wedge \exists i \in \{1..n\} \bullet (i, j) \in A\}$.

Similarly, X^{A+} and Y^{A+} are the subsequences of X^A and Y^A that are *paired* and *matched* by the alignment:

$X^{A+} = \{x_i \mid x_i \in X \wedge \exists j \in \{1..m\} \bullet (i, j) \in A \wedge x_i \approx y_j\}$

$Y^{A+} = \{y_j \mid y_j \in Y \wedge \exists i \in \{1..n\} \bullet (i, j) \in A \wedge x_i \approx y_j\}$.

Then, the *mismatched* subsequences are defined as follows: $X^{A-} = X^A - X^{A+}$, and $Y^{A-} = Y^A - Y^{A+}$.

Finally, the *gap* subsequences are those whose elements that are matched with gaps: $G_X^A = X - X^A$, and $G_Y^A = Y - Y^A$.

Definition II.4 (Degree of matching)**.** Given an alignment A between two traces X and Y, we can define its *degree of matching* (m_A) with respect to each trace as follows:

$m_A(X) = \#X^{A+}/\#X$

$m_A(Y) = \#Y^{A+}/\#Y$

where "#" denotes the number of elements of a sequence. Both degrees are real numbers between 0 and 1.

Definition II.5 (Sequence of consecutive gaps)**.** Given an aligment A, a subset S of G_A is a *sequence of consecutive gaps* (SCG) if the elements of S fulfill the following properties:

- $\forall i \in \{i_m..i_M\} \bullet (i, 0) \in S$
- $\forall j \in \{j_m..j_M\} \bullet (0, j) \in S$
- $i_m > 1 \wedge j_m > 1 \Rightarrow (i_m - 1, j_m - 1) \in A$
- $i_M < n \wedge j_M < m \Rightarrow (i_M + 1, j_M + 1) \in A$

where $i_m = \min\{i \mid (i, 0) \in S\}$, $i_M = \max\{i \mid (i, 0) \in S\}$, $j_m = \min\{j \mid (0, j) \in S\}$, and $j_M = \max\{j \mid (0, j) \in S\}$.

The length of a sequence of consecutive gaps S is given by the size of the set S.

Intuitively, $[i_m, i_M]$ defines the range of indexes of X in S, and $[j_m, j_M]$ defines the range of indexes of Y in S. The first two conditions require that the sequence S is bounded by paired elements, i.e., the pair of A just before the first element of S is either a match or a mismatch, and the pair of A right after the last element of S is either a match or a mismatch in A, too. The last two conditions require that the pairs of S have consecutive indexes, i.e., there are no paired elements in all the range of indexes of S.

Example 1. Suppose that $X = \{A, B, C, D, E, F, G, H\}$, $Y = \{A, M, E, H\}$ and that the alignment A is defined as follows:

$A = \{(1,1), (2,0), (3,0), (0,2), (4,0), (5,3), (6,0), (7,0), (8,4)\}$

In this case, only three points have been matched, namely $\{(1,1), (5,3), (8,4)\}$, and the rest have been identified as gaps, i.e., $G_A = \{(2,0), (3,0), (0,2), (4,0), (6,0), (7,0)\}$.

Then, we can identify two sequences of consecutive gaps: $S_1 = \{(2,0),(3,0),(0,2),(4,0)\}$ and $S_2 = \{(6,0),(7,0)\}$, with respective lengths of 4 and 2.

Should we prefer to represent the alignment by the elements of X and Y, the sequences of consecutive gaps correspond to $S_1 = \{(B,-),(C,-),(-,M),(D,-)\}$ and $S_2 = \{(F,-),(G,-)\}$, while the matched sequence is $\{A,E,H\}$.

Note that, in general, representing alignments using the elements of the sequences to be matched is not appropriate, especially since they may contain repeated elements. This would make it impossible to distinguish the specific matches and thus the respective gap sequences. This is why the alignments are always specified by the indexes of the elements of the respective sequences and not by the elements themselves.

Note as well that three alternative alignments could have been defined:

$A' = \{(1,1),(2,2),(3,0),(4,0),(5,3),(6,0),(7,0),(8,4)\}$

$A'' = \{(1,1),(2,0),(3,2),(4,0),(5,3),(6,0),(7,0),(8,4)\}$

$A''' = \{(1,1),(2,0),(3,0),(4,2),(5,3),(6,0),(7,0),(8,4)\}$

They pair respectively element M in Y with B, C and D in X, identifying them as mismatches. Their corresponding SCGs are the following:

$S'_1 = \{(3,0),(4,0)\}$ and $S'_2 = \{(6,0),(7,0)\}$, both of length 2.

$S''_1 = \{(2,0)\}$, $S''_2 = \{(4,0)\}$ and $S''_3 = \{(6,0),(7,0)\}$, of lengths 1, 1 and 2.

$S'''_1 = \{(2,0),(3,0)\}$ and $S'''_2 = \{(6,0),(7,0)\}$, both of length 2.

Thus, depending on how the alignment decides to pair the elements and thus determine the corresponding gaps, the length of the SCGs may vary. This is further illustrated in the next example.

Example 2. Suppose now that $X = \{A,A,A,B,B,B,B,B\}$ and $Y = \{A,B\}$. In this case, depending on the strategy of the alignment algorithm to determine the gaps, we can have alignments with rather different sequences of consecutive gaps. For example, one alignment matches the elements in the extremes of sequence X, obtaining only one SCG of length 6. Another alignment pairs x_4 with y_1 and x_5 with y_2, obtaining two SCGs of lengths 2 and 4. An alternative alignment has three SCGs of lengths 1, 3 and 2. There are 12 possible alignments, each one with resulting SCGs of different lengths.

The decision of whether we prefer more SCGs of shorter length or fewer but longer SCGs depends on the strategy of the alignment algorithm to open a gap or continue exploring the chain.

III Alignment Algorithms

In order to align traces, we explored existing solutions for sequence alignment. Sequence alignment is a widely studied problem in various fields, such as Bioinformatics, where efficient algorithms have been developed for aligning DNA and other biological sequences. For our specific task of aligning behavioral traces, we employed a combination of well-established techniques. Firstly, we utilized the renowned global alignment algorithm for biological sequences called Needleman-Wunsch (NDW) [1] as the foundation for our algorithm development. Additionally, we incorporated a set of techniques derived from the popular BLAST algorithm [2], which is widely used to search databases of biological sequences for statistically significant similarities. These techniques will be elaborated on in the subsequent sections.

A. The Needleman-Wunsch Algorithm

The *Needleman-Wunsch algorithm* [1] is a global alignment algorithm that finds the optimal alignment of two sequences of characters, A and B. It is implemented using Dynamic Programming techniques, i.e., it breaks down the problem of comparing sequences into smaller problems (comparing sets of subsequences) to find the optimal solution to the global problem, and uses a similarity matrix to re-use calculations. The Needleman–Wunsch algorithm is widely used for optimal global alignment, particularly when the quality of the global alignment is of the utmost importance. The algorithm assigns a score to every possible alignment, and tries to find one of the possible alignments having the highest score—there may be no unique optimal alignment.

Algorithm 1 Algorithm to calculate the similarity matrix for the Needleman-Wunsch algorithm [1]

```
1:  M[0][0] ← 0
2:  c ← 1
3:  while c < M[0].size do                          ▷ Fill the first row of the matrix
4:      M[0][c] ← M[0][c − 1] + scoreIns
5:      c ← c + 1
6:  end while
7:  r ← 1
8:  while r < M.size do
9:      M[r][0] ← M[r − 1][0] + scoreDel             ▷ Fill first column
10:     while c < M[0].size do
11:         ins ← M[r][c − 1] + scoreIns
12:         sub ← M[r − 1][c − 1] + scoreSub(A[r − 1], B[c − 1])
13:         del ← M[r − 1][c] + scoreDel
14:         M[r][c] ← max{ins, sub, del}
15:         c ← c + 1
16:     end while
17: end while
```

To select the optimal one, the algorithm requires a scoring scheme, which is defined in terms of rewards and penalties assigned to the possible results when comparing two characters c_A and c_B, belonging to sequences A and B, respectively. These outcomes are:

- **Match.** The characters are the same.
- **Mismatch.** The characters are not the same but are aligned, which is considered a mismatch.
- **InsDel** (or **Gap**). The characters are not the same and the best alignment involves one letter paired to a gap (represented by "–") in the other sequence.

There are two prevailing scoring schemas used in these situations:

- **Constant scoring schema.** Each of the results is assigned a score, for example a reward of $+1$ for a Match, 0 for a Mismatch, and a penalty of -1 for a Gap. This scheme gives priority to mismatches over gaps.
- **Frequency-based scoring schema.** Each possible character comparison includes a specific set of scores for each result. These scores are usually recorded in an input matrix that the algorithm uses for the alignment. Input matrices are the most common scoring system used in Bioinformatics since some proteins are more common than others [3]. The values of the scores are obtained empirically.

The alignment is performed using a similarity matrix which includes sequence A along the vertical axis and sequence B along the horizontal axis. Each cell in the matrix represents a pairing between a character of each sequence. To fill the matrix, the algorithm maximizes in each cell the score, choosing between the three outcomes. The pseudocode is listed in Algorithm 1, in which M is the similarity matrix, and A and B are the character sequences to align. The $scoreSub$ is evaluated after comparing the characters and determining whether it is a match or a mismatch (line 12).

Once the matrix is filled, the algorithm traces back from the bottom right by choosing the maximal scores. The result of the algorithm is the alignment that maximizes the scores.

There are different scoring schemas depending on how we would like the alignment to work. For example, we could use a reward of $+1$ for a Match; 0 for a Mismatch; and a penalty of -1 for a Gap. This scheme gives priority to mismatches over gaps. Another schema uses a reward of $+1$ for matches and penalties of -1 and -2 for mismatches and gaps, resp. This schema corresponds to the edit distance between the two strings [4]. The lower the alignment score the larger the edit distance.

B. BLAST

Basic Local Alignment Search Tool (BLAST) [2] is a publicly available algorithm that compares biological sequences against a library of sequences and provides a list of the most similar ones given a similarity threshold. BLAST is based on a *local*

alignment algorithm for biological sequences based on Dynamic Programming. *Global* alignment algorithms, such as Needleman-Wunsch [1], attempt to align all the elements of the sequences. In contrast, *local* alignment algorithms compare dissimilar sequences to find regions with high similarity levels, such as the Smith-Waterman algorithm [5], on which BLAST is based. However, these algorithms find the maximum scoring alignment between sequences, while BLAST, using heuristics, only reports statistically significant alignments based on a threshold. This way, the algorithm does not explore the entire search space between two sequences, alleviating computational costs. However, it cannot guarantee optimal solutions.

To avoid traversing the entire search space, it uses three heuristic phases to refine potential high-scoring pairings.

- **1) Seeding.** The user establishes a specific word length, w. Then the algorithm looks for alignments of that length whose score is as large as a threshold T. These matches are the seeds for possible alignments.
  - During the seeding phase, the algorithm uses **Low complexity regions**, which are subsequences of little biological interest as they are common in many other sequences, but which produce high scores. To avoid seeding in such regions, they are soft-masked, including them in the alignment in the extension phase but not in the seeding phase.
- **2) Extension.** Once the search space is seeded, the algorithm generates new alignments by extending the individual seeds, increasing the word length progressively. To determine when to stop extending, a threshold X decides how much the alignment score may drop from the last maximum.
- **3) Evaluation.** Once all the seeds are extended, the alignments are evaluated to decide which ones are statistically significant. To determine this significance, the algorithm establishes a score threshold S, which allows sorting alignments into low- and high-scoring.

C. Affine Gap

The result of any alignment algorithm may include gaps. In the Needleman-Wunsch algorithm (see Section III-A), the penalty for a gap is set as a constant value for all the alignments. This produces alignments with sequences in which gaps and matches alternate. However, this approach may not work well when the expected alignments have long gaps, instead of many small sequences of alternating gaps and matches. For instance, alignments with longer gaps are more meaningful when trying to detect anomaly sequences [6]. To better model this phenomenon, alignment algorithms such as BLAST typically apply an evaluation gap penalty system named *affine gap* which imposes a higher penalty for opening a new gap than for extending an existing gap. If P_{op} is the penalty for opening a gap, and P_{ex} is the penalty for extending one, the total penalty of opening a gap of length L would be $P_{op} + P_{ex} * (L - 1)$.

These penalties are incorporated into the scores of the Dynamic Programming algorithm so that the optimal alignment is built considering them. The use of affine gaps requires keeping track of the possibilities of opening or continuing a gap in either of the sequences for every pair of characters. This introduces the need for four additional matrices, in addition to the similarity matrix used in any Dynamic Programming algorithm, hence introducing more time and space complexity to the algorithm.

To implement an affine gap penalty [6], we need to keep track of the possibility of opening or continuing a gap in either of the sequences for every pair of characters. For this, we need four matrices, one per sequence for the opening or continuing a gap decision, one to evaluate matches and mismatches, and one which selects the maximum value of the three, which are respectively:

$H_A(i,j)$– maximum score of the alignment of A and B where $A[i]$ is aligned with a gap
$H_B(i,j)$– maximum score of the alignment of A and B where $B[i]$ is aligned with a gap
$H_M(i,j)$– maximum score of the alignment of S and B where $A[i]$ and $B[i]$ are aligned
$H(i,j)$ – maximum score of the alignment of A and B

The initial conditions to fill in the first row and column of the matrix are the following:

$$H(i,0) = H(0,j) = 0$$
$$H_M(i,0) = H_M(0,j) = -\infty$$
$$H_A(i,0) = H_B(0,j) = p_{op} + p_{ex}$$
$$H_A(0,j) = H_B(i,0) = -\infty$$

The formulas to fill the dynamic programming matrices are as follows:

$$H(i,j) = max\{H_N(i,j), H_A(i,j), H_B(i,j)\}$$

$$H_N(i,j) = H(i-1,j-1) + equals(A[i], B[j])$$
$$H_A(i,j) = max\{H(i-1,j) - (p_{op} + p_{ex}), H_A(i-1,j), -p_{ex}\}$$
$$H_B(i,j) = max\{H(i-1,j) - (p_{op} + p_{ex}), H_B(i-1,j), -p_{ex}\}$$

Once the matrix is filled using this scheme, the back-tracing process is the same as in the NDW algorithm: Select the maximal score at the bottom-right of the matrix, i.e., $max\{H(i,j)\}$, and trace back choosing the maximal scores.

The values for the penalties of opening and extending a gap for BLAST are obtained empirically and usually depend on the frequency scoring matrix used for the alignment [7]. In general, the algorithm works well when the penalty for opening a gap (P_{op}) is between 10 and 15 times the penalty of continuing it (P_{ex}) [2].

IV Fidelity Metrics

Metric IV.1 (Percentage of matched snapshots)**.** Given an alignment A between two traces X and Y, we define its *percentage of matched snapshots* ($\%m_A$) as follows:

$$\%m_A^+ = \#X^{A+}/max(\#X, \#Y) * 100$$

Note that $\#X^{A+} = \#Y^{A+}$ since there is an equal number of elements from each set in the alignment, as each aligned element has its counterpart in the other set. Additionally, it is worth noting that typically the traces should have a similar number of elements. However, in the event of a discrepancy, we will evaluate the percentage based on the trace with the greater number of elements. This means that both systems pass through the same states at the same moments, generating a set of equivalent snapshots with a one-to-one coincidence

Metric IV.2 (Frèchet distance)**.** The Frèchet distance $F(X^{A+}, Y^{A+})$ between two aligned traces X^{A+} and Y^{A+} is defined as the infimum over all reparametrizations χ and ψ of the maximum over all $t \in [0,1]$ of the distance between $X^{A+}(\chi(t))$ and $Y^{A+}(\psi(t))$. It is generally explained through this example: Imagine a person who walks from one end of a trajectory X to the other end, being their position $X^{A+}(\chi(t))$; likewise, a dog walks from one end of its trajectory Y to the other end, being its position $Y^{A+}(\psi(t))$, with the person holding the dog by a leash. The Frèchet distance between both is the minimum leash length needed to walk the dog following their trajectories. Formally, assuming that $d(X^{A+}(\chi(t)), Y^{A+}(\psi(t)))$ is a distance measure such as the Euclidean or Manhattan's distance, the Fréchet distance is defined as follows:

$$F(X^{A+}, Y^{A+}) = \inf_{\chi,\psi} \max_{t\in[0,1]} \left\{ d(X^{A+}(\chi(t)), Y^{A+}(\psi(t))) \right\}$$

Metric IV.3 (Euclidean mean distance)**.** Given two already aligned traces $X^{A+} = \{x_1, .., x_n\}$ and $Y^{A+} = \{y_1, .., y_n\}$, i.e., the algorithm finds a sequence of n equivalent pairs of points from both sequences, where the elements x_i and y_i describe the state of each system at step i. Then, assuming that $d(p,q)$ is a distance measure between a pair of points p and q (e.g., the Euclidean or the Manhattan distance), we can compute the average distance between the two traces, $d(X^{A+}, Y^{A+})$, as well as its sample standard deviation, $s(X^{A+}, Y^{A+})$, as follows:

$$d(X^{A+}, Y^{A+}) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, y_i)$$

$$s(X^{A+}, Y^{A+}) = \sqrt{\frac{1}{n-1} \left(\sum_{i=1}^{n} d(x_i, y_i)^2 - n \cdot d(X^{A+}, Y^{A+})^2 \right)}$$

It measures the average distance between all the paired snapshots that are aligned. Unlike the Frèchet distance, which only detects changes in the maximum value, this metric provides an overview of the alignment quality, considering both the mean and the standard deviation, making it less sensitive to outliers.

Additionally, we have some complementary metrics to reason about the possible causes of any misalignments.

Metric IV.4 (Percentage of mismatches and gaps)**.** Given an alignment A between two traces X and Y, we define its *percentage of mismatches and gaps* ($\%m_A^-$ and $\%m_A^G$, resp.) as follows:

$\%m_A^- = \#X^{A-}/max(\#X, \#Y) * 100$

$\%m_A^G = 100 - (\%m_A^+(X) + \%m_A^-(X))$

Note that $\#X^{A-} = \#Y^{A-}$ because when a mismatch occurs, an element from each of the two traces is aligned, resulting in an equal number of elements from each trace in the list of mismatches. A higher percentage of gaps than mismatches may indicate that there is a delay between behaviors, whereas a high percentage of mismatches could mean a temporary deviation of the behavior.

Metric IV.5 (Number of individual gaps)**.** Given an alignment A between two traces X and Y, we define its *number of individual gaps* ($\#G^A$) with respect to each trace as follows:

$\#G_X^A = \#(X - X^A)$

$\#G_Y^A = \#(Y - Y^A)$

Metric IV.6 (Number of groups of consecutive gaps)**.** Given an alignment A between two traces X and Y, and the set of consecutive gaps S^A over the alignment, as presented in Definition II.5, we define the *number of groups of consecutive gaps* as $\#S^A$.

Metric IV.7 (Mean length of the gaps)**.** Given an alignment A between two traces X and Y, and the set of consecutive gaps $S^A = \{S_1^A, ..., S_k^A\}$ over the alignment, as presented in Definition II.5, we define the *mean length of the gaps* ($\overline{S^A}$) with respect to each trace as follows:

$\overline{S^A} = \frac{1}{k} \sum_{i=1}^{k} \#S_i^A$

These metrics help analyze the distribution of gaps in the trace and provide insights into the potential differences between them without the need for visualizing the alignment. For example, if there is a small number of very long gaps, it may indicate a delay in the behavior of one of the two systems. Similarly, if there are many clusters of very short gaps, it may be a symptom of stuttering behavior in one of the twins.

References

[1] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990. [Online]. Available: https://blast.ncbi.nlm.nih.gov/

[3] S. Henikoff and J. G. Henikoff, "Amino acid substitution matrices from protein blocks." *Proceedings of the National Academy of Sciences*, vol. 89, no. 22, pp. 10 915–10 919, 1992.

[4] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[5] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022283681900875

[6] S. F. Altschul, B. W. Erickson, and H. Leung, *Local Alignment (with Affine Gap Weights)*. Boston, MA: Springer US, 2008, pp. 459–461.

[7] I. Korf, M. Yandell, and J. A. Bedell, *BLAST - an essential guide to the basic local alignment search tool*. O'Reilly, 2003.