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

A General Concepts

1 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 of 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 Digital Twin Systems (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.

2 Definitions

Definition A.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 A.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 A.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 A.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 A.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.

3 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) [2] as the foundation for our algorithm development. Additionally, we incorporated a set of techniques derived from the popular BLAST algorithm [3], which is widely used to search databases of biological sequences for statistically significant similarities. These techniques will be elaborated on in the subsequent sections.

3.1 Sequence Alignment: Global and local alignments

In the literature, sequence alignment algorithms are divided into two categories: *global* and *local alignment*.

The *global alignment* approach forces an alignment of the entire length of the longest sequence and attempts to align every character in every sequence [4]. This approach is usually applied when sequences are expected to be similar and approximately the same size. A general global alignment algorithm is the Needleman-Wunsch (NDW) [2].

For example, let us have the sequences $S_1 = \{G,T,C,G,A,C,G,C,A\}$ and $S_2 = \{G,A,T,T,A,C,A\}$. A possible global alignment between both, using the NDW algorithm, is $A_0 = \{(G,G),(T,A),(C,T),(G,T),(A,A),(C,C),(G,-),(C,-),(A,A)\}$. This alignment represents the most optimal alignment of all elements in both sequences, comprising 3 mismatches, 2 gaps, and 4 matches. Further details will be explored in the

following section.

The *local alignment* approach identifies regions of similar subsequences within long sequences [4]. This approach is usually applied in the case of dissimilar sequences that are expected to contain similar subsequences. A general local alignment algorithm is the Smith-Waterman [5]. The result of this algorithm, in contrast to the NDW, does not produce an alignment covering the entire length of the two sequences but rather a list of the most similar subsequences between them.

Let us perform the alignment of sequences S_1 and S_2 using the Smith-Waterman algorithm. The result includes the alignments alignments with the highest score, being aligned subsequences from S_1 and S_2, respectively.

$$
\begin{aligned}
A_1 &= \{(G,G),(A,T),(T,C),(T,G),(A,A),(C,C)\} \\
A_2 &= \{(G,G),(A,T),(T,C),(T,G),(A,A),(C,C),(A,G)\} \\
A_3 &= \{(G,G),(A,A),(T,-),(T,C),(A,G),(C,C),(A,A)\} \\
A_4 &= \{(G,G),(A,A),(T,C),(T,-),(A,G),(C,C),(A,A)\} \\
A_5 &= \{(G,G),(A,A),(T,C),(T,G),(A,-),(C,C),(A,A)\}
\end{aligned}
$$

3.2 The Needleman-Wunsch Algorithm

a) Overview

The *Needleman-Wunsch algorithm* (NDW) is a global alignment algorithm that finds *"the largest number of amino acids in one protein that can be matched with those of a second protein allowing for all possible interruptions in either of the sequences"* [2]. The two amino acid sequences are represented by 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 score 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 utmost importance, i.e., when we want to detect regions of similar content considering the entire length of the sequence. The algorithm assigns a score to every possible alignment and tries to find the alignments having the highest score. There may be no unique optimal alignment since it depends on the algorithm scoring scheme. This 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.
- **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 nucleotides or amino acids mutate or are substituted easier by others [6]. The values of the scores are obtained empirically.

The alignment is performed using a score 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 to fill the dynamic programming matrix is listed in Algorithm 1, in which M is the score matrix, and A and B are the character sequences to align. The r function 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 tracks back from the bottom right by choosing the maximal scores. The result of the algorithm is the alignment that maximizes the scores.

Algorithm 1 Algorithm to calculate the similarity matrix for the Needleman-Wunsch algorithm [2]

$M[0][0] \leftarrow 0$
$c \leftarrow 1$
while $c < M[0].size$ **do** ▷ Fill the first row of the matrix
    $M[0][c] \leftarrow M[0][c-1] + gap$
    $c \leftarrow c + 1$
end while
$r \leftarrow 1$
while $r < M.size$ **do**
    $M[r][0] \leftarrow M[r-1][0] + gap$ ▷ Fill first column
    while $c < M[0].size$ **do**
        $ins \leftarrow M[r][c-1] + gap$
        $sub \leftarrow M[r-1][c-1] + r(A[r-1], B[c-1])$
        $del \leftarrow M[r-1][c] + gap$
        $M[r][c] \leftarrow max\{ins, sub, del\}$
        $c \leftarrow c + 1$
    end while
end while

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. If we apply this schema to align the sequences $S_1 = \{G, T, C, G, A, C, G, C, A\}$ and $S_2 = \{G, A, T, T, A, C, A\}$. The resulting alignment is the same as in the previous section:

$$A_0 = \{(G, G), (T, A), (C, T), (G, T), (A, A), (C, C), (G, -), (C, -), (A, A)\}$$

However, if we use a reward of $+1$ for a Match; -1 for a Mismatch; and a penalty of 0 for a Gap, we have a scheme that prioritizes gaps over mismatches and the resulting alignment would be:

$$A'_0 = \{(G, G), (-, A), (T, T), (-, T), (C, -), (G, -), (A, A), (C, C), (G, -), (C, -), (A, A)\}$$

As we can see in the alignment A'_0, if the penalty for adding a gap is zero, we obtain optimal alignments that include a gap for each of the characters that cannot be aligned with an equivalent in the other sequence. This configuration makes it highly unlikely for the reward of including a mismatch to outweigh the penalty in the global score. This configuration is as valid as the previous one, but it is not as widely used in Bioinformatics since the presence of mismatches could be interpreted as a point of mutations [7].

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, i.e., the Levehnstein distance, between the two strings [8]. The lower the alignment score the larger the edit distance.

b) Bellman's Principle of Optimality

The Needleman-Wunsch algorithm must fulfill Bellman's Principle of Optimality to employ a dynamic programming approach. **Bellman's Principle of Optimality** can be summarized as follows: *"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy concerning the state resulting from the first decision."* [9]

This principle implies that the optimal solution to the problem contains the optimal solutions of all the reduced versions of the same problem that can be solved in a nested manner. This also implies that, by knowing the optimal solution to these reduced versions of the problem, we can deduce the optimal decisions needed to reach that solution.

The Needleman-Wunsch algorithm can be formulated using nested subproblems: the alignment of subsequences in the input strings. This ensures an optimal substructure, making it possible to resolve the problem by applying a dynamic programming approach.

(A) Characterize the Structure of an Optimal Solution. Given two sequences of characters $C = \{c_i\}_{i=1}^{n}$ and $D = \{d_j\}_{j=1}^{m}$ and a substitution matrix $m : C \times D \rightarrow \mathbf{Z}$ that determines the similarity score between two characters, an alignment A satisfies the properties described in Definition A.1.

Each time a gap is added to the alignment, a constant gap penalty, $g < 0$, is added to the score. Our goal is to obtain the alignment that pairs the most similar characters according to the substitution matrix m. Then, the solution, $A = \{a_1, \ldots, a_p\}$, can be defined as a set of pairs, $a_i = (i_{a_i}, j_{a_i})$.

Assuming these definitions, the function that defines the score of a given pair is the following:

$$r(a_i) = \begin{cases} m(a_i), & \text{if } i_{a_i} > 0 \wedge j_{a_i} > 0 \\ g & \text{if } i_{a_i} = 0 \vee j_{a_i} = 0 \end{cases}$$

To solve the complete problem, the algorithm must maximize $R(A) = \sum_{i=1}^{p} r(a_i)$, where p represents the number of pairs in the alignment A, i.e., $p = |A|$. A condition regarding the cardinality of A is that, in the worst case, if all the characters in both sequences were aligned with gaps, the cardinality of A would be, at most, the sum of the lengths of the aligned sequences C and D, i.e., $|A| \leq |C| + |D|$.

(B) Proof of the Optimal Substructure. Let us assume that we have two sequences of characters C and D and an optimal alignment of these two sequences named A^*. If our problem follows an optimal substructure, it must fulfill the condition that ${A^*}_{2 \rightarrow p} = \{a_2^*, ..., a_p^*\}$ is the optimal solution to align all the characters except those included in the pair a_1^*.

To arrive at a contradiction and demonstrate the optimality of A^*, let us assume that A^* is not the optimal solution, and there exists another alignment $A' = \{a'_2, \ldots, a'_p\}$ that satisfies:

$$|A'| \leq |C \setminus \{i_{a_1^*}\}| + |D \setminus \{j_{a_1^*}\}| \tag{Eq. A.3.1}$$

$$\sum_{i=2}^{p} r(a'_i) > \sum_{i=2}^{p} r(a_i^*) \tag{Eq. A.3.2}$$

If this were true, we could build $A'' = \{a_1^*, a'_2, ..., a'_p\}$ that fulfills that:

$$|A'| + |\{a_1^*\}| \leq |C \setminus \{i_{a_1^*}\}| + |\{i_{a_1^*}\}| + |D \setminus \{j_{a_1^*}\}| + |\{j_{a_1^*}\}| \tag{Eq. A.3.3}$$

$$\sum_{i=2}^{p} r(a''_i) = r(a_1^*) + \sum_{i=2}^{p} r(a'_i) > \sum_{i=1}^{p} r(a_i^*) \tag{Eq. A.3.4}$$

We reach a contradiction since we assumed that A^* is the optimal solution to our problem and, therefore, it cannot exist a suboptimal solution with a higher score.

(C) Recursive definition using Bellman's Equation. This proves that the NDW algorithm follows the optimal substructure and we can express it using Bellman's Equation in a recursive manner:

$$N_{i,j} = max \begin{cases} N_{i-1,j-1} + r(c_i, d_j) & \text{Match or Mismatch} \\ N_{i,j-1} + g, & \text{Gap in sequence } D \\ N_{i-1,j} + g, & \text{Gap in sequence } C \end{cases} \tag{Eq. A.3.5}$$

Each of the decisions that the algorithm makes to solve the problem will consist in determining if it aligns two characters or adds a gap in either of the sequences C or D. Based on this, we derive the formula in Eq. A.3.5, in which we assume a matrix of dimensions (m, n) with the characters of C indexed by i to the characters of D indexed by j.

3.3 BLAST

Basic Local Alignment Search Tool (BLAST) [3] 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 the dynamic programming local alignment algorithm Smith-Waterman [5]. There are only minor differences between Smith-Waterman algorithm and NDW. In both cases, characters are individually compared using a constant scoring scheme or a substitution matrix, resulting in a score for each pair of characters. By summing these individual scores and applying gap penalties, each aligned subsequence is assigned a similarity score. The higher this score is, the more relevant the alignment, as it will contain more matches and fewer gaps and mismatches. To filter out irrelevant alignments, BLAST allows users to set a numerical threshold for this score. This threshold is used to consider only highly relevant alignments of subsequences during the search.

Smith-Waterman traverses the entire search space to identify the highest-scoring alignments. However, this approach becomes impractical when searching for matches in databases containing thousands of sequences. To tackle this problem, BLAST employs heuristics to report alignments that are statistically significant given a similarity threshold. This strategy avoids exploring the entire search space between two sequences, reducing computational costs. Nevertheless, 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 masks **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.

3.4 Affine Gap

The result of any alignment algorithm may include gaps. In the Needleman-Wunsch algorithm (see Section A.3.2), 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 [10]. 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 three additional matrices in addition to the score matrix used in any Dynamic Programming algorithm, hence introducing more time and space complexity to the algorithm.

To implement an affine gap penalty [10], 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 Bellman Equations for this extended problem are as follows:

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

Once the matrix is filled using this scheme, the back-tracking 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 track 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 [11]. 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}) [3].

4 Fidelity Metrics

This section introduces a set of metrics used to compare alignments between two sequences and offers precise definitions for these metrics.

Metric A.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:

$$\%m_A^+ = \#X^{A+}/max(\#X, \#Y) * 100 \qquad \text{(Eq. A.4.1)}$$

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 A.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:

$$F(X^{A+}, Y^{A+}) = \inf_{\chi,\psi} \max_{t\in[0,1]} \left\{ d(X^{A+}(\chi(t)), Y^{A+}(\psi(t))) \right\} \qquad \text{(Eq. A.4.2)}$$

Metric A.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:

$$d(X^{A+}, Y^{A+}) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, y_i) \quad \text{(Eq. A.4.3)}$$

$$s(X^{A+}, Y^{A+}) = \sqrt{\frac{1}{n-1} \left(\sum_{i=1}^{n} d(x_i, y_i)^2 - n \cdot d(X^{A+}, Y^{A+})^2 \right)} \quad \text{(Eq. A.4.4)}$$

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 A.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:

$$\%m_A^- = \#X^{A-} / max(\#X, \#Y) * 100 \quad \text{(Eq. A.4.5)}$$

$$\%m_A^G = 100 - (\%m_A^+(X) + \%m_A^-(X)) \quad \text{(Eq. A.4.6)}$$

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 A.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:

$$\#G_X^A = \#(X - X^A) \quad \text{(Eq. A.4.7)}$$

$$\#G_Y^A = \#(Y - Y^A) \quad \text{(Eq. A.4.8)}$$

Metric A.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 A.5, we define the *number of groups of consecutive gaps* as $\#S^A$.

Metric A.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 A.5, we define the *mean length of the gaps* ($\overline{S^A}$) with respect to each trace as follows:

$$\overline{S^A} = \frac{1}{k} \sum_{i=1}^{k} \#S_i^A \quad \text{(Eq. A.4.9)}$$

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.

5 Snapshot-based Needleman-Wunsch: *Our trace Alignment Algorithm*

In this section, we will provide a comprehensive description of how we adapted certain algorithms and techniques discussed in the background sections and examine their impact on the Bellman Optimality principle.

5.1 Needleman-Wunsch algorithm adapted for Trace Alignment

Let us consider the recursive definition of the NDW algorithm as presented in Section A.3.2b.

$$N_{i,j} = max \begin{cases} N_{i-1,j-1} + r(c_i, d_j) & \text{Match or Mismatch} \\ N_{i,j-1} + g, & \text{Gap in sequence } D \\ N_{i-1,j} + g, & \text{Gap in sequence } C \end{cases} \quad \text{(Eq. A.5.1)}$$

In Eq. A.5.1, N represents the score matrix used to calculate the global solution through dynamic programming. The variable g stands for the constant penalty for introducing a gap, and $r(c_i, d_j)$ is the reward function, which assigns a value based on the similarity between characters c_i and d_j.

This reward can be constant in the case of a *constant scoring scheme* (e.g., match +1, and mismatch -1), or it can be derived from a *substitution matrix*, such as *BLOSUM 62* [12]. Substitution matrices are empirically generated by analyzing alignments of protein sequences that are nearly identical and calculating the probability of one amino acid being substituted by another. As a result, a substitution matrix assigns an integer reward for each pair of amino acids.

Our approach adjusts the reward function $r(c_i, d_j)$ to a similarity function between two snapshots, as described in Algorithm 2. This function produces real values within the interval (0, 1] for a match and 0 for a mismatch between snapshots. The function's output represents an average of the level of similarity in each of the attributes of the snapshot. We consider three cases based on attribute types, but it can be extended to any number of types as long as the comparison returns a value within the range [0, 1], which reflects the similarity between the values of a given attribute. The cases considered are the following:

- **Numerical values** (line 3). The algorithm calculates the absolute difference between the attributes values. For each numerical value, there is a predefined *Maximum Acceptable Distance (MAD)* between values, representing the maximum distance to categorize them as a match. The closer the values are to each other with respect to the MAD, the closer the reward approaches its maximum value of 1 (line 6). If this distance is exceeded, the comparison is regarded as a mismatch (line 8).
- **Boolean values** (line 12). If the values are the same, we consider it a match and add the maximum reward to the average; otherwise, we add 0.
- **String values** (line 15). The *equals* can be any method that measures the similarity between strings, such as the Levenshtein distance [8].

Therefore, this similarity function only takes into account the values of the input snapshot's attributes, providing a value based solely on them without considering the values of other snapshots. This preserves the independence between subproblems, which is a prerequisite for Bellman's Optimality Principle. Essentially, we replace the substitution matrix applied to amino acids with an equivalent process that assesses the similarity between snapshots while maintaining the original approach and ensuring optimality. With this structure, the optimality proof would be analogous to the one presented in Section A.3.2b.

Algorithm 2 Similarity function between two snapshots

```
1: i ← 1; result ← 0
2: while i ≤ k do
3:     if isNumerical(s_A.a_i) then
4:         dif ← abs(s_A.a_i − s_B.a_i)
5:         if dif < MAD then
6:             result ← result + (1 − dif/MAD)
7:         else
8:             result ← 0                          ▷ Mismatch if MAD exceeded
9:             break
10:        end if
11:    end if
12:    if isBoolean(s_A.a_i) & s_A.a_i = s_B.a_i then
13:        result ← result + 1
14:    end if
15:    if isString(s_A.a_i) then
16:        result ← result + equals(s_A.a_i, s_B.a_i)
17:    end if
18:    i ← i + 1
19: end while
20: return result/k
```

5.2 BLAST techniques applied: Low-complexity regions

In BLAST, during the seeding phase where the algorithm seeks relevant partial matches, certain common amino acid sequences are masked to prevent their inclusion in the search as high-scoring matches [11]. These sequences are found in many proteins and do not characterize any specific family, making them a poor starting point for a search. By masking these high-scoring matches, the algorithm can focus on more characteristic regions, enabling more precise and efficient searches within the database.

During our experiments, we observed that some global alignments of snapshots could face a similar issue. Certain sequences composed of common high-scoring but irrelevant snapshots could outweigh the reward associated with behavior-characteristic regions. Consequently, these alignments might treat relevant regions as gaps rather than matching them, especially when these regions could not yield a reward as high as other regions.

The elevator case study that we used to validate the algorithm encountered this issue. To assess the simulator's accuracy in determining the elevator's movement, we focused on acceleration, as it allowed us to derive speed and position. In many segments of the trace, the elevator did not exhibit significant acceleration, with values close to 0 (less than $0.05m/s^2$). These segments generated rewards very close to 1, which outweighed the rewards produced by the acceleration curves. These acceleration curves constituted a smaller set of snapshots with significantly lower rewards. Consequently, the algorithm considered these curves as gaps in its efforts to maximize the reward. However, these curves were among the most relevant aspects for evaluating the precision of acceleration, whereas the others were only of interest for assessing delay.

To reduce the impact of these segments on the overall score, we opted to mask these regions by decreasing their returned score. Therefore, in modifying the algorithm as outlined in Algorithm 2 to the one in Algorithm 3, we introduce a constant denoted as *LOW*. If both snapshots belong to a low-complexity area, the score is divided by twice the constant value *LOW* (line 19). If either of the snapshots is part of a low-complexity area, the reward is divided by the value of *LOW* (line 21).

As in the adaptation of the NDW algorithm presented in the previous section, this modification only changes the reward returned by the similarity function, which solely depends on the values of the two snapshots being evaluated. The *isLowComplexity* function will be implemented for each system, depending on the parts of the trace that the user intends to mask. For instance, in the case of the elevator, only values below $0.05m/s^2$ are categorized as low-complexity areas. This decision should be made by a domain expert based on the results, and the value of *LOW* can be determined through fine-tuning of the algorithm's parameters or by measuring the proportion of low-complexity snapshots relative to significant behavior.

This design maintains the independence of subproblems to adhere to the Bellman Optimality Principle, ensuring an optimal solution through a dynamic programming algorithm.

Algorithm 3 Similarity function between two snapshots including Low-Complexity Areas

$i \leftarrow 1;\ result \leftarrow 0$
while $i \leq k$ **do**
  if $isNumerical(s_A.a_i)$ **then**
    $dif \leftarrow abs(s_A.a_i - s_B.a_i)$
    if $dif < MAD$ **then**
      $result \leftarrow result + (1 - dif/MAD)$
    else
      $result \leftarrow 0$ ▷ Mismatch if *MAD* exceeded
      break
    end if
  end if
  if $isBoolean(s_A.a_i)\ \&\ s_A.a_i = s_B.a_i$ **then**
    $result \leftarrow result + 1$
  else
    result $\leftarrow 0$
    break
  end if
  if $isString(s_A.a_i)$ **then**
    $result \leftarrow result + equals(s_A.a_i, s_B.a_i)$
  else
    result $\leftarrow 0$
    break
  end if
  if $isLowComplexity(s_A.a_i)\ \&\ isLowComplexity(s_B.a_i)$ **then**
    $result \leftarrow result/2LOW$
  else if $isLowComplexity(s_A.a_i) \vee isLowComplexity(s_B.a_i)$ **then**
    $result \leftarrow result/LOW$
  end if
  $i \leftarrow i + 1$
end while
return $result/k$

5.3 BLAST techniques applied: Affine Gap

This technique is not exclusive to BLAST, but it is also applied to other sequence alignment algorithms like Smith-Waterman [5] or Needleman-Wunsch [2]. This technique modifies the Bellman Equation of the original algorithm by adding three more equations, which will impose three more matrices to calculate the optimal alignment, resulting in an increase in memory consumption. These equations address the problem in three circumstances assuming we are aligning two traces $A = \{a_i\}_{i=1}^{n}$ and $B = \{b_j\}_{j=1}^{m}$: align a_i with a gap in sequence B, align b_j with a gap in sequence A, and align a snapshot from A with a snapshot of B. Then, the algorithm selects the case that maximizes the score and incorporates it into the main matrix. Subsequently, backtracking is performed on this matrix to determine the optimal alignment.

The method is applied as outlined in the literature and detailed in Section A.3.4. The only modification made is in the similarity function, which still preserves the subproblem structure as explained in Section A.5.1. Therefore, it can be concluded that all the adaptations applied to the algorithm maintain the suboptimal structure and adhere to the Bellman Principle of Optimality, as presented in Section A.3.2b.

B Variants and state-of-the-art comparisons

1 Introduction

In this document, we present a comparison of our algorithm, named *Snapshot-based Needleman-Wunsch (SbNDW)*, with its variants (Section B.2) and other state-of-the-art sequence alignment algorithms (Dynamic Time Warping, Longest Common Subsequence).

2 Snapshots-based Needleman-Wunsch's variants comparison

Verifiability: The alignments performed in this section are available at https://github.com/atenearesearchgroup/fidelity-measure-for-dts/blob/main/src/resources/output/lift/variants_comparison

2.1 Overview

The **Snapshot-based Needleman-Wunsch (SbNDW)** algorithm incorporates a series of optimizations compared to the original Needleman-Wunsch algorithm [2]. Primarily, we extended the algorithm to align not only sequences of characters, but also sequences of snapshots. We introduced a similarity function that returns a value between 0 and 1. This function guides the algorithm to determine the optimal alignment by maximizing the amount of similarity values returned by this function. Consequently, the algorithm identifies the sequence of pairs of snapshots that exhibit the highest degree of similarity.

To align a sequence of snapshots effectively, the algorithm takes into account the potential inclusion of gaps between pairs of snapshots. This implies that certain snapshots in one sequence may lack a counterpart in the other, so they are aligned with a so-called *gap*. Whenever a gap is included, a constant penalty is incorporated into the score. However, in certain scenarios, it is undesirable to have gaps and matches alternating, as this does not contribute to identifying anomalous areas [10]. To address this concern, we have incorporated the option to use the ***Affine Gap*** schema in our algorithm. This scoring scheme imposes a higher penalty for opening a gap than for extending one. This preference for longer gaps over shorter ones aids in better defining areas of anomalous behaviors, delays, or misalignments.

When employing sequence alignment algorithms such as BLAST [3], there may be instances where certain common subsequences yield high scores in the alignment but do not contribute to characterizing system behavior. In our work, we name these subsequences as ***Low-Complexity Regions***. These areas are prioritized in the matching process, potentially hampering the identification of the behavior of interest. In BLAST, these regions are masked during the seeding phase to mitigate the influence of this high-scoring phenomenon on the alignments. In our case, we propose masking these areas to prevent the algorithm from aligning them over regions containing relevant behavioral information. While not always necessary, this feature proves beneficial in systems that may incorporate low-complexity areas, thereby allowing the algorithm to focus on the pertinent information. To mask these areas, we incorporate a *Low-Complexity Area Weight (LCAW)* that is applied to the similarity reward, reducing its value.

In this section, we will compare four variants of the algorithm using the elevator case study in the (4-0-4) scenario. In this scenario, the elevator goes down from the 4th floor to the ground floor and then back up again. We recorded ten measurements of the Physical Twin (PT) for this scenario. The Digital Twin (DT) intended to be used as a counterpart is a high-fidelity commercial simulator named *Elevate* [13]. This simulator is deterministic, providing only one trace for this scenario. We will analyze the impact on the resulting alignments and metrics when including the LCAW and the Affine Gap. Additionally, a performance analysis will evaluate the effects of these optimizations on computational cost and complexity.

The input parameters were fine-tuned for the Elevator Case Study as outlined in [14]. Thus, in this analysis, we will utilize the following configuration:

Parameter	Value
Maximum Acceptable Distance (MAD)	0.15
Penalty opening a gap (P_{op})	-1
Penalty extending a gap (P_{ex})	-0.1
Low Complexity Areas Weight (LCAW)	0.005

The variants analyzed will be the following:

Variant	Affine Gap	LCAW
Base	No	No
LCAW	No	Yes
Affine Gap	Yes	No
LCAW + Affine Gap	Yes	Yes

2.2 Results

In [14], we showed that the alignments conducted on these measurements, using the recommended configuration for the *LCAW + Affine Gap variant*, exhibited a high level of fidelity. Specifically, in scenario (4-0-4), the average values across all measurements were 94.3% matched snapshots (MS), $0.11\ m/s^2$ Frèchet distance (FD), and $0.05\ m/s^2$ Euclidean distance (ED).

In the following subsections, we illustrate how the alignments are affected by the individual and combined use of the Affine Gap and the LCAW.

a) Low-Complexity Regions Effect

After running alignments with the optimal configuration on the ten measurements of the (4-0-4) scenario, it is observed that two out of ten measurements' alignments are affected by the presence of *Low-Complexity areas* (Figures 1 and 2).

Let us start with the analysis of Figure 1a, which corresponds to measurement number 4. In this instance, the PT starts the movement delayed about 10 seconds with respect to the DT. Our algorithm is expected to align the regions that exhibit similar sequences of states. A proper alignment allows us to (i) assess the level of similarity between states and (ii) identify and label sequences of behavior that do not match any sequence in the other trace.

When we apply this to the data in Figure 1a, we notice that the first two acceleration sequences (representing the movement from the fourth floor to the ground floor) are not aligned with their corresponding sequences in the other trace. However, this misalignment is not observed in the other two acceleration curves, which are aligned with their respective counterparts. An alignment with these characteristics could provide (i) inaccurate metrics, as comparison misses the alignment of characteristic behavior, and (ii) labels are absent parts that, in fact, are included in the other trace.

The problem is analogous in Figure 2a, but the missing aligned curves are the latter two, representing the transition from the ground to the fourth floor. In this case, these curves occurred earlier than anticipated in relation to the DT. Similar to Figure 1a, when applying the base algorithm, these two curves remain unaligned. Consequently, the resulting alignment might be misinterpreted as insufficient similarity between the two accelerated curves, which is not the case.

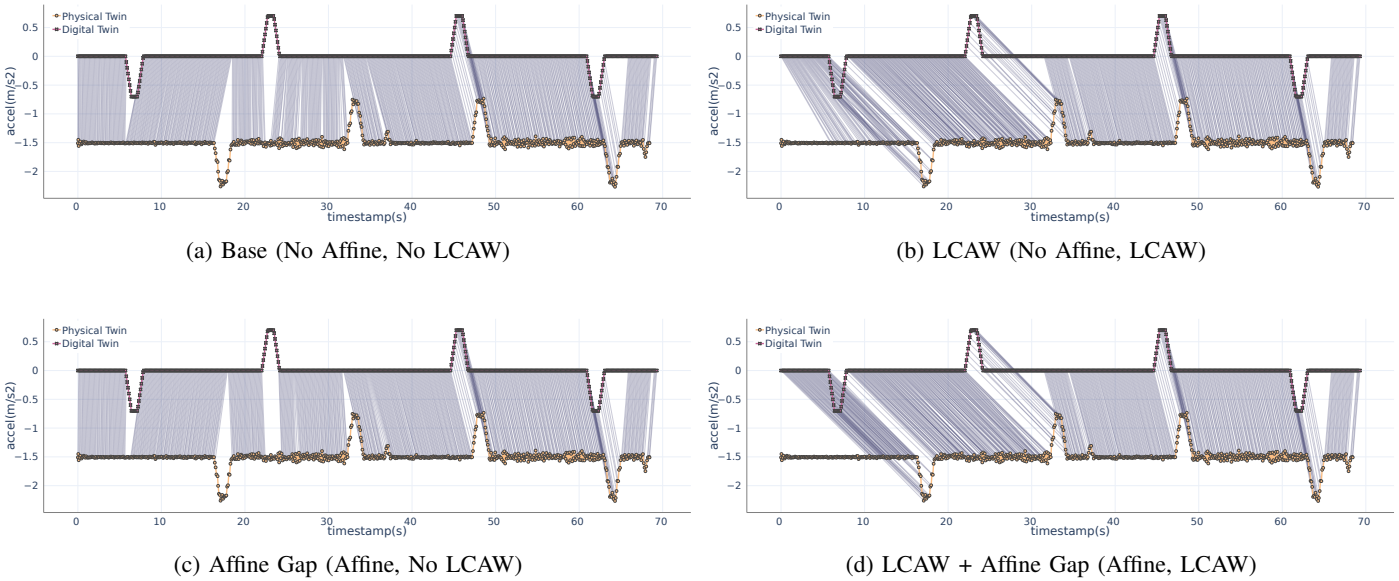
The problem in both cases is due to the algorithm's reward scheme. As outlined in the paper and in [15], the similarity function returns a value in the range (0, 1]. The greater similarity between the two snapshots corresponds to a higher returned value. In the case of snapshots with numerical attributes, the similarity is evaluated as the absolute difference between them.

If we return to our example, we realize that the reward scheme will benefit the alignments included in the zero-acceleration regions over the ones in the acceleration curves. For example, consider an acceleration value of $0\ m/s^2$ for the DT and a noised value of $0.005\ m/s^2$ within noise areas of the zero-acceleration areas. Their difference is 0.005, normalized with respect to the MAD at $0.15\ m/s^2$. The reward value is $1 - (0.005/0.15) = 0.97$. In contrast, let us examine two potentially similar snapshots of the acceleration curves: $0.7\ m/s^2$ and $0.76\ m/s^2$. Their difference is 0.06, and if we apply the same reasoning, we obtain the following reward: $1 - (0.06/0.15) = 0.6$

This indicates that the algorithm consistently assigns significantly higher rewards for matches within the zero-acceleration areas, i.e., the Low-Complexity regions. While the algorithm's aim is to align similar areas, this aim prioritizes these regions over the behavior of interest: the acceleration curves. Consequently, returning imprecise alignments.

The majority of snapshots within the acceleration curve exhibit a similar difference. These values undergo rapid changes compared to those in the zero-acceleration areas, and the system captures snapshots periodically, not accounting for every value the system passes through, but rather specific snapshots. Consequently, the algorithm needs to accommodate certain tolerance in the comparisons.

Following these considerations, we introduced a weight to reduce the reward of alignments within these areas, prompting the algorithm to prioritize matches in relevant areas. The application of LCAW is depicted in Figures 1b and 1d, and Figures 2b



(a) Base (No Affine, No LCAW)

(b) LCAW (No Affine, LCAW)

(c) Affine Gap (Affine, No LCAW)

(d) LCAW + Affine Gap (Affine, LCAW)

Fig. 1: Alignments for Scenario (4-0-4), Measurement 04

and 2d, where the acceleration curves are properly aligned.

Let us analyze the alignment in Figure 1b. Previously unaligned gaps within the acceleration curves are now matched with three sets of zero-acceleration areas across both traces, in sequential order:

1) A starting delay is represented as a group of gaps at the beginning of the PT's trace.
2) After arriving at the fourth floor, the PT starts moving again before the DT, which waits longer, resulting in a series of gaps in the DT's trace.
3) Due to window size constraints and the initial delay, the PT trace ends with fewer snapshots. A group of snapshots following the last DT acceleration represents this set of missing data.

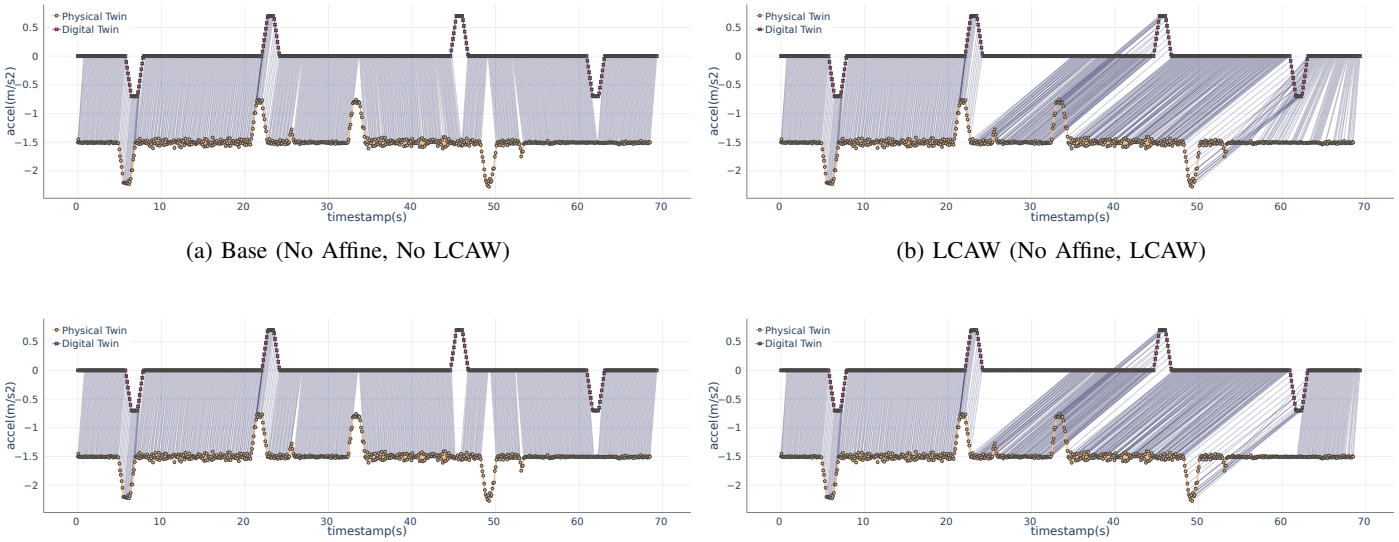
The analysis would be similar for alignments depicted in Figure 2b:

1) The DT's delay with respect to the PT, occurring after reaching the fourth floor and initiating movement towards the ground floor, is represented by a series of gaps in the DT's trace.
2) As the PT initiated its movement earlier than the DT, there is a set of gaps after the PT reaches and halts at the fourth floor. These gaps are also due to window size constraints and are absent in the other trace.

The average metrics for the ten measurements of scenario (4-0-4) are detailed in Table I. When analyzing the impact of LCAW on the percentage of matched snapshots (%MS), it shows an average reduction of 2%, while the Frèchet and Euclidean distances were minimally affected. However, as presented in this section, misaligned snapshots influence the categorization of sequences. For instance, examining the metrics for Measurement 04 in Figure 1, there's a considerable impact: %MS decreases from 89% without LCAW to 83% after its application. This difference is significant. Similarly, Measurement 09 in Figure 2 experiences a decline from 92.5% to 82%, potentially impacting the system's perceived reliability.

b) Affine Gap Effect

The primary objective of employing Affine Gap is to minimize the occurrence of non-consecutive gaps in alignments, prioritizing the identification of anomalous or missing areas. To evaluate this effect, we could analyze Figures 1, 2 and 3, focusing on subfigures (a) and (b) without Affine Gap, compared to subfigures (c) and (d) where it is applied. In Figures 1 and 2, as we zoom in, the alternation between gaps and matches in the alignment is reduced and substituted by continuous areas of gaps and matches.



(a) Base (No Affine, No LCAW)

(b) LCAW (No Affine, LCAW)

(c) Affine Gap (Affine, No LCAW)

(d) LCAW + Affine Gap (Affine, LCAW)

Fig. 2: Alignments for Scenario (4-0-4), Measurement 09

To illustrate the significance of this improvement, let us consider Figure 3. At the beginning of the alignment, there is a delay in the PT with respect to the DT. This delay would be represented as a sequence of gaps. However, if we do not apply Affine Gap (Subfigure 3a), this delay is depicted as a combination of gaps and matches, making the interpretation more challenging. Implementing LCAW (Subfigure 3b) alleviates this issue, yet an alternating sequence of matches and gaps persists. As we apply Affine Gap in Subfigures 3d and 3c, this issue is effectively solved, presenting a clear sequence of gaps at the start of the sequence.

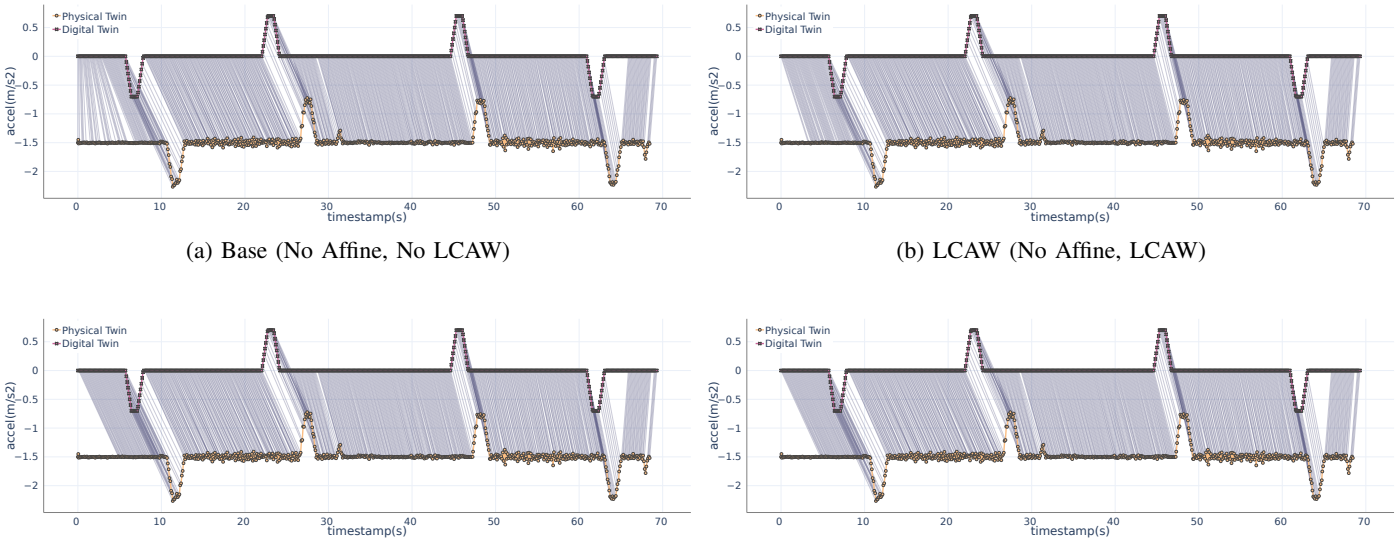
When examining the aggregated statistics across all measurements, as depicted in Table I, it becomes apparent that applying the Affine Gap does not have a considerable impact on fidelity statistics. However, a closer look at the gap statistics reveals a substantial increase in the average gap length, going from 5 to 11, without a rise in the overall number of gaps. Nonetheless, these gaps are now consolidated into 10 groups instead of the previous 20+, effectively halving the non-aligned areas within the alignment. This consolidation leads to a clearer interpretation of the data.

TABLE I: Average fidelity metrics for the ten measurements scenario (4-0-4).

Variant	Frèchet	Euclidean	% Matches	% Mismatches	% Gaps	Gap length	#Gaps	#Groups of Gaps
Base (No Affine, No LCAW)	0.124	0.022	93.438	0.446	6.116	3.596	82	24
LCAW (No Affine, LCAW)	0.126	0.023	91.797	0.417	7.785	4.792	105	22
Affine (Affine, No LCAW)	0.127	0.023	93.481	0.935	5.584	7.935	75	10
LCAW + Affine Gap	0.129	0.024	91.884	0.417	7.699	11.741	104	9

c) Low-Complexity Areas Weight + Affine Gap Effect

After analyzing the separate effects of the optimizations, we conclude that their combined application provides two key benefits: (i) achieving a more accurate alignment applying LCAW, preventing the alignment of Low-Complexity Regions over crucial behavioral patterns, and (ii) generating longer, more contextually relevant groups of gaps by applying Affine Gap, which contributes significantly to better interpretation. Applying both together, as could be perceived in the Figures and the aggregated data, enhances the benefits that they provide individually, obtaining higher quality alignments.



(a) Base (No Affine, No LCAW)

(b) LCAW (No Affine, LCAW)

(c) Affine Gap (Affine, No LCAW)

(d) LCAW + Affine Gap (Affine, LCAW)

Fig. 3: Alignments for Scenario (4-0-4), Measurement 05

3 Comparison with Dynamic Time Warping

Verifiability: The alignments performed in this section are available at https://github.com/atenearesearchgroup/fidelity-measure-for-dts/blob/main/src/resources/output/lift/state_of_the_art_comparison

3.1 Algorithm overview

Dynamic time warping (DTW) [16] is a Dynamic Programming algorithm that measures the similarity between two temporal sequences, which may vary in speed. As our SbNDW algorithm, it can be applied to any sequence alignment problem in which a similarity distance between the elements can be established. It computes an optimal alignment between two sequences: the one that minimizes pair distances.

The Bellman Equation for DTW is:

$$D(i,j) = \text{dist}(s_i, t_j) + \min\{D(i-1,j), D(i,j-1), D(i-1,j-1)\} \quad \text{(Eq. B.3.1)}$$

This equation represents the recursive definition of the DTW distance calculation. It considers $D(i,j)$ as the accumulated cost at the position (i,j) in the alignment matrix. The function dist(s_i, t_j) denotes the similarity distance between the elements s_i and t_j, the i-th element of sequence s, and the j-th element of sequence t. The minimum function identifies the minimum cost path to reach (i,j), considering the three possible previous positions.

Unlike NDW, DTW allows one-to-many and many-to-one matches because it assumes that one sequence is a time-warped version of the other. In this regard, DTW accounts for scenarios where the target sequence is either stretched (one-to-many), condensed (many-to-one), or unwarped (one-to-one) concerning the source sequence. Consequently, DTW does not support the concept of gaps, while NDW explicitly considers and penalizes gaps by assigning a penalty.

DTW minimizes pair distances, with the resulting value serving as a representation of similarity between the sequences. A smaller distance signifies a higher similarity between the two traces.

3.2 Theoretical comparison with SbNDW

NDW and DTW are dynamic programming algorithms designed to find the optimal alignment between two sequences of elements. Their goal is to find the sequence of pairs of the most similar elements between the two sequences. Our proposed variant of NDW, called SbNDW, modifies the NDW algorithm to enable the alignment of any kind of elements, assuming that a distance measure between these elements can be defined.

Our proposal works under the assumption that both sequences must align one-to-one with the other. This condition holds within the context of a Digital Twin System (DTS) where two replicas work simultaneously and report their state with identical periodicity. If both systems exhibit similar behavior simultaneously, their traces can be aligned one-to-one. DTW works not only under this assumption but also considers speed variations in which the many-to-one alignments could fit.

Since DTW allows one-to-many alignments, it lacks the support for gaps. Consequently, the algorithm aligns all elements, including those exhibiting low similarity. In doing so, it pairs these elements with the most similar ones within a convenient section of the sequence, trying to minimize any collateral penalty for the alignment of adjacent elements. These alignments increase the resulting distance, indicating a lower similarity between the two traces compared to others. However, these dissimilar pairs are not explicitly identified or labeled and are included as any other pair in the resulting alignment. In contrast, SbNDW categorizes these points as either gaps or mismatches, thereby simplifying the user's task of identifying differences between the traces or pinpointing anomalies.

Another primary limitation of DTW is its tendency to over-stretch or over-compress traces. This occurs when a point in one trace aligns with multiple points in the other sequence, leading to an excessive emphasis on a single point in an attempt to minimize the distance. As a result, the distance results obtained may be unfaithful and not accurately represent the actual similarity between the traces. The inadequacy of this approach has been highlighted by studies in the field. Various proposals include methods to mitigate this issue [17].

3.3 Statistical Comparison SbNDW

To demonstrate the DTW limitations mentioned in the previous section, we align the measurements of the (4-3-2-1-0-1-2-3-4) scenario. In this scenario, the elevator travels from the fourth floor to the ground floor, stopping at each floor, and then goes back up. This is the scenario that we will use for all the comparisons against other proposals, as it provides a longer trace for better analysis of performance and results.

The configuration for the alignment of the SbNDW algorithm is the one presented in B.2.1. DTW is executed without any additional configuration. The distance measurements between the snapshots are the absolute difference between the acceleration values.

The results of the alignments are in Figure 4. If we analyze the elevator trajectory in any of the figures, we observe a delay in the PT compared to the DT. In Subfigures 4c and 4b, this delay is represented by a sequence of gaps at the beginning of the alignment. The delay could be determined by multiplying the number of gaps by the snapshot-taking period. The rest of the snapshots are aligned one-to-one with occasional inclusion of gaps. The fidelity metrics estimate approximately 95% of matched snapshots (%MS), Frèched distance (FD) of $0.11m/s^2$, and Euclidean distance (ED) of $0.02m/s^2$. Reporting a high level of fidelity.

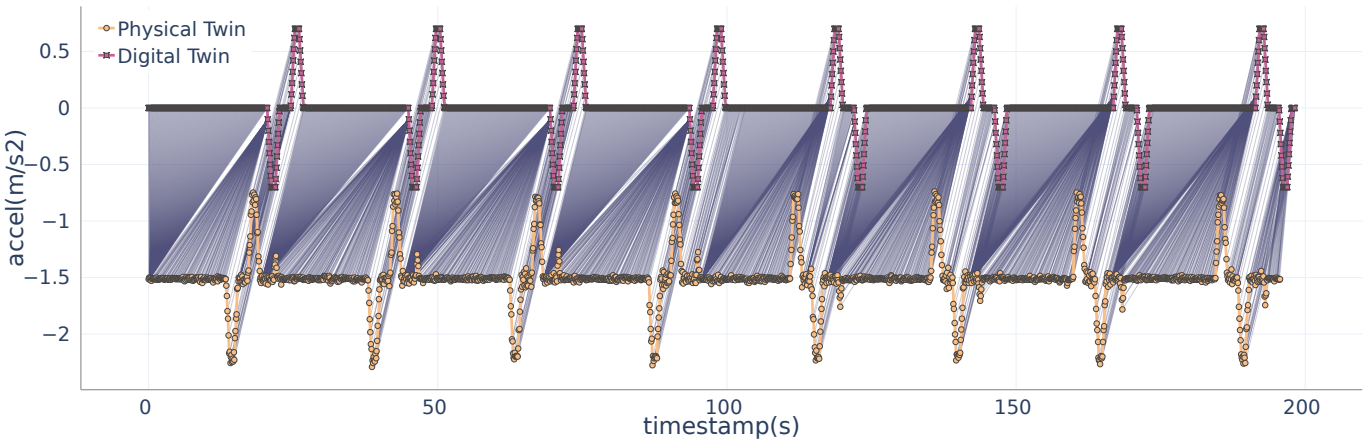
Let us consider now Subfigure 4a. When analyzing the alignment from left to right, it is clear how the algorithm is employing over-stretching and over-compression. This results in one snapshot being assigned to multiple snapshots. For example, the DT zero-acceleration values before the first curve are aligned with one value in the PT that is very close to zero. Similarly, the values of the noise zero-acceleration zone of the PT are aligned with one value that is close to the acceleration curve in the DT. This pattern is repeated throughout the alignment.

In this alignment, the over-stretching and over-compressing phenomena are prevalent, as presented in the statistical results in Table II. As a result, one point is often aligned with as many as 190 other snapshots in the DT and 206 snapshots in the PT, which is approximately 9% of the entire trace. On average, each snapshot is aligned at least 40 times, resulting in more than 1500 one-to-many alignments.

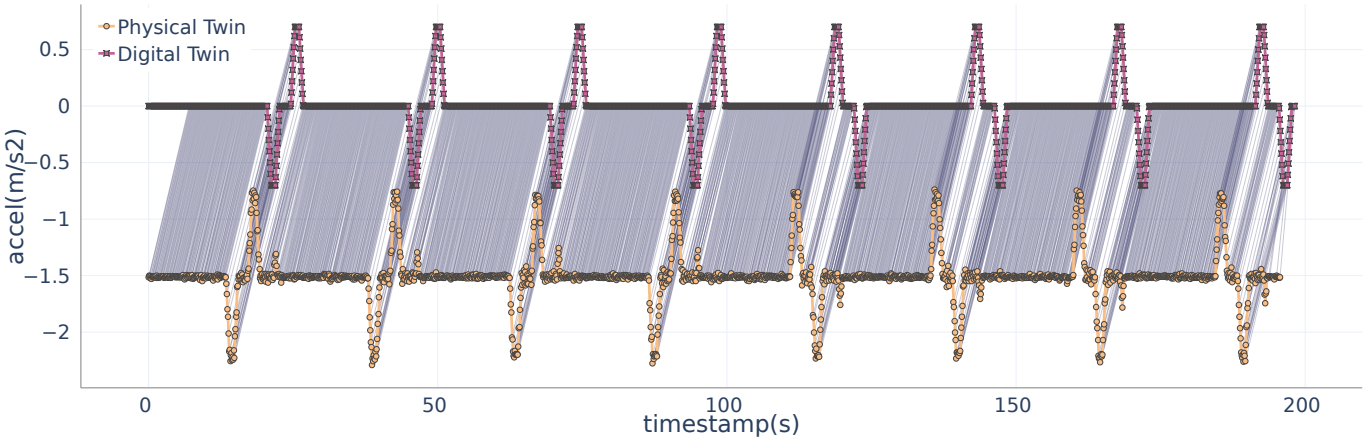
The similarity score obtained during the alignment of a trace consisting of 1983 snapshots using DTW is $39.45m/s^2$, which is relatively low. However, it is important to note that this value takes into account the over-stretching and over-compression phenomena, which do not compare the supposedly simultaneous behavior. The distance calculated in the zero-acceleration curves is not representative of reality as it is only aligned with one point. Despite analyzing the resulting score value and the alignment in Subfigure 4a, it is difficult to determine the origin of any imprecision or differences between the traces.

4 Comparison with the proposal by Lugaresi et al. [1]

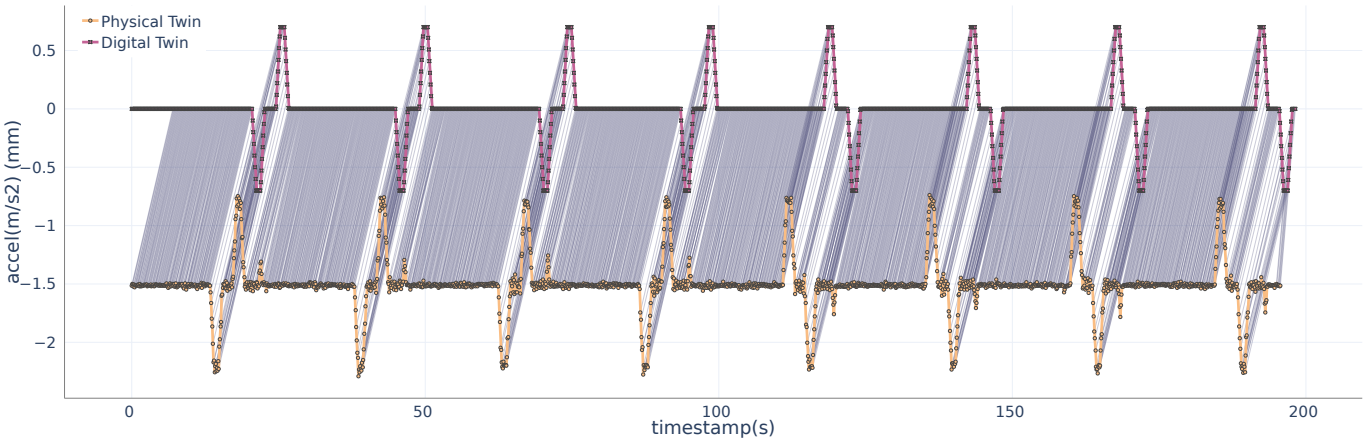
Verifiability: The alignments performed in this section are available at https://github.com/atenearesearchgroup/fidelity-measure-for-dts/blob/main/src/resources/output/lift/state_of_the_art_comparison



(a) Dynamic Time Warping



(b) LCAW (No Affine, LCAW)



(c) Affine Gap + LCAW

Fig. 4: Alignments for Scenario (4-3-2-1-0-1-2-3-4)

TABLE II: Dynamic Time Warping statistics for scenario (4-3-2-1-0-1-2-3-4)

Metric	DT	PT
Max times a snapshot is aligned	190	206
# Snapshots aligned more than once	41	34
Avg times a snapshot is aligned more than once	37.80	45.44
Std times a snapshot is aligned more than once	69.58	75.77
Total one-to-many alignments	1510	1509
Trace length	1983	
Similarity score	39.45 m/s2	

4.1 Overview

In 2023, Lugaresi et. al [1] published a paper entitled "*Online Validation of digital twins for manufacturing systems*." The algorithm presented in that paper shared the same principles as our proposal: It aimed to compare the behavior of Digital Twins by evaluating the sequence of discrete behavioral traces using dynamic programming algorithms. Additionally, the paper defined fidelity metrics to measure the accuracy of their results.

Despite both solutions target a similar problem, there are some significant differences between the proposals. Firstly, while Lugaresi et. al [1] compare the traces at run-time, we compare the two traces offline. In any case, the moment at which the two traces are compared is not critical for comparing the behavior of the two algorithms. Secondly, our work is designed to analyze any type of trace, from raw information to event analysis, whereas their work is more specific, and centers around events and analysis of KPIs, which is located at a higher level of abstraction. Our proposal allows for the comparison of snapshots of any type, as long as a distance measure between 0 and 1 is defined. As validation results, we provide the alignment between the traces, which can be used to interpret the presence of gaps and mismatches, as well as fidelity metrics (%MS, FD, ED). Their proposal includes the validation at two levels: event-level validation and KPIs / performance-level validation, which are discussed below.

a) Event-level validation

At this level, a check is performed to determine if the events generated by the twins are comparable. Events are identified by a unique string identifier and a timestamp. The **Longest Common Subsequence (LCSS)** algorithm is used to evaluate these traces. This dynamic programming algorithm returns the highest number of matching elements, in the same sequential order, between the two traces.

The Bellman equation for this algorithm is the following:

$$L(a_i, b_j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ L(a_{i-1}, b_{j-1}) + 1 & \text{if } d(a_i, b_j) = 0 \wedge d(t_{a,[i]}, t_{b,[j]}) < \delta \\ \max\{L(a_{i-1}, b_j), L(a_i, b_{j-1})\} & \text{otherwise} \end{cases} \quad \text{(Eq. B.4.1)}$$

In this formula, $L(a_i, b_j)$ is the accumulated score of the given subsequence. If the characters, or events in this case, are equivalent ($d(a_i, b_j) = 0$), then a +1 is added to the global score of such subsequence. In this case, Lugaresi et al. added another consideration: $d(t_{a,[i]}, t_{b,[j]}) < \delta$. They introduced a parameter called δ, which restricts the maximum time difference between two events to be considered equivalent. This allows to define a time window for equivalence and avoid aligning events that are too far apart.

The algorithm produces a numerical value representing the length of the LCSS. The authors developed a fidelity metric for this result, which ranges from 0 to 1, normalizing the value. The fidelity metric indicates the faithfulness of the sequence of events. The closer the value is to 1, the more faithful the sequence is. For that, they divide the LCSS value by the length of the shorter trace.

They could have access to the included elements in the LCSS, but they do not reconstruct the optimal solution by performing backtracking (this part is not present in their code). The optimal solution, that is, the optimal subsequence, could help them

reason about which parts do not fit and therefore are not included in the alignment. The only value they use from the algorithm output is the score, which is the length of the optimal subsequence.

b) KPIs / Performance-level validation

At this level, the performance of the system is evaluated to ensure that it matches the results obtained in the DT experiments. The KPIs are obtained in the form of numerical sequences, which are analyzed using two algorithms: **Modified Longest Common Subsequence (mLCSS)** and **Dynamic Time Warping (DTW)**.

The first algorithm, **Modified Longest Common Subsequence (mLCSS)** follows the same Bellman Equation as the one presented in the previous subsection:

$$L(a_i, b_j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ L(a_{i-1}, b_{j-1}) + 1 & \text{if } d(a_i, b_j) < \epsilon \\ \max\{L(a_{i-1}, b_j), L(a_i, b_{j-1})\} & \text{otherwise} \end{cases} \quad \text{(Eq. B.4.2)}$$

The only difference is that, when dealing with numerical values, it is necessary to determine when two values can be considered equivalent. This is achieved by introducing the value ϵ as the maximum distance for aligning two KPIs. If the distance is below this threshold, the pair can be considered part of the LCSS.

As mentioned in the preceding section, the algorithm produces a numerical value that represents the length of the LCSS. This value is then normalized by dividing it by the length of the shortest sequence. The result ranges from 0 to 1 and determines the degree of fidelity between the traces. As in the previous subsection, the LCSS is not obtained for posterior analysis.

The second algorithm included by Lugaresi et al. in their proposal is **Dynamic Time Warping (DTW)**, which was introduced in the previous section B.3. The objective of the authors is to develop a metric that ranges between 0 and 1. To achieve this, they start by normalizing all the input sequence values by dividing them by the maximum value of the two sequences. Then, they perform the alignment process and re-normalize the outcome ($D_{[m,n]}(a, b)$) by dividing the distance obtained by the length of the longest sequence. Finally, the metric is calculated by subtracting this value from 1.

$$\Phi_{DTW}(a, b) = 1 - \frac{D_{[m,n]}(a, b)}{max(m, n)} \quad \text{(Eq. B.4.3)}$$

If we align two identical sequences, the numerator will be 1 because the outcome from DTW will be 0. On the other hand, if the maximum value and minimum value of any two elements of the KPI sequences are 1 and 0, respectively, the maximum distance between the furthest values will be 1. If all the values are at this distance, the resulting alignment distance will be m. This means that the denominator and the numerator will be the same, which is 1. As a result, the outcome will be 0.

This approach suffers from all the limitations of DTW, mentioned in the previous section B.3. Additionally, it solely relies on the output score without conducting any analysis of the output alignment. As a result, this approach may potentially overlook results that have been affected by over-stretching or over-compression.

An additional limitation could be found in the normalization process. Let us consider an example of two traces of 5 elements each:

$$A = \{0.05, 0.05, 0.05, 0.05, 0.05\} \qquad B = \{0.01, 0.02, 0.03, 0.04, 0.05\}$$

We normalized the traces by dividing both traces by their maximum value (0.05):

$$\overline{A} = \{1, 1, 1, 1, 1\} \qquad \overline{B} = \{0.2, 0.4, 0.6, 0.8, 1\}$$

If we assume that the returned alignment consists of aligning the elements at the same position, we obtain the following set of distances:

$$D(A, B) = D(\{0.8, 0.6, 0.4, 0.2, 0\}) = 2$$

Applying the metric to this output distance, we obtain:

$$\Phi_{DTW}(A, B) = 1 - \frac{2}{5} = 0.6$$

Now, let us compare this situation with a similar alignment, including a trace B' that has an outlier. The normalized traces would be the same since the maximum value is 1.

$$A = \overline{A} = \{0.05, 0.05, 0.05, 0.05, 0.05\} \qquad B' = \overline{B'} = \{0.01, 0.02, 0.03, 0.04, \mathbf{1}\}$$

The current alignment was expected to perform worse because an outlier was included instead of a perfect match. However, the actual result shows a better alignment. This is because of the normalization process. Since the normalization factor is one instead of a value smaller than 1, the smaller distance values are not increased as in the previous example. In this case, the smaller distance values have a smaller weight in the distance result, masking the inclusion of the outlier.

$$D(A, B) = D(\{0.04, 0.03, 0.02, 0.01, 0.95\}) = 1.05$$

In the second example, when we add up the distance values, we get 1.05 instead of 2. This indicates that the DTW algorithm considers the traces to be closer, which is not true. With that distance, the metric returns a value of 0.79, which is higher than 0.6. This suggests that the traces are more similar, but upon examining the traces, it is found to be incorrect.

$$\Phi_{DTW}(A, B) = 1 - \frac{1.05}{5} = 0.79$$

4.2 Theoretical comparison with SbNDW

The approach proposed by Lugaresi et al. [1] has an advantage over SbNDW in that it offers three metrics within the range from 0 to 1, without considering units. Our proposal also includes three metrics, but two of them involve units as they take into account the distance between aligned pairs. It is important for domain experts to interpret these metrics to determine the system's validity.

However, both proposals ultimately require the consideration of domain knowledge. In this particular proposal, the authors take into account the values of δ and ϵ, which serve as upper limits to determine whether two sequence elements are equivalent when applying LCSS. These values are similar to our Maximum Acceptable Distance (MAD).

They propose analyzing aggregated metrics: events and KPIs. This assumes that the system can identify states in a continuous environment, which may not be trivial in some cases. It is important to note that when dealing with KPIs, aggregating values may mask anomalous events. This is because an outlier may take some time to affect certain metrics, resulting in an inefficient monitoring system. Therefore, it is important to carefully consider which metrics to use when analyzing KPIs. Our proposal is applicable to these situations as we align raw data, which is unaffected by aggregation and does not require event identification.

The proposal incorporates a metric based on DTW but does not consider over-stretching and over-compression, which could potentially affect the accuracy of their results. Furthermore, the normalization process applied to the metric may produce incomparable results, as explained in the previous section.

4.3 Statistical comparison with SbNDW

In this section, we will compare the SbNDW algorithm with the three metrics proposed by Lugaresi et al. We will use the scenario (4-3-2-1-0-1-2-3-4) and the optimal configuration for the SbNDW algorithm as introduced in section B.2.1.

The output of our approach for this scenario is in Subfigures 4c and 4b.

a) Event-level validation - Longest Common Subsequence

For the event validation, we modified our raw trace including four types of events: "Down," "Up," "Brake," and "Arrival". Each event was assigned to a specific timestamp which corresponds to the acceleration changes during the elevator's operation. Specifically, "Up" and "Down" events were assigned to timestamps when the acceleration increased, "Brake" was assigned when the elevator began to reduce acceleration, and "Arrival" was assigned when the elevator stopped moving. In more complex scenarios, the user may need to automatically infer this information.

Labeling our original trace with these events, the new trace includes 24 events. Modifying the value of δ, we obtain the results in Table III. The alignment in Figure 5a shows that all events are included, and no behavior is missing, but they are delayed in

time. Increasing the value of δ leads to the maximum score of 1. By gradually increasing δ and analyzing the transitions, we can observe that most of the traces are delayed between 7 and 7.7 seconds.

We previously performed an analysis using SbNDW, which allowed us to derive this conclusion. However, this analysis only showed that with a certain δ value, we can obtain a certain similarity result, without providing any information about the possible reasons behind this result. This is one of the main limitations of this analysis.

TABLE III: Event-validation using LCSS statistics for scenario (4-3-2-1-0-1-2-3-4). Trace length of 24 events.

δ	normalized score	score / LCSS length
6.9	0.00	0
7	0.29	7
7.1	0.29	7
7.2	0.42	10
7.3	0.54	13
7.4	0.75	18
7.5	0.92	22
7.6	0.96	23
7.7	1.00	24

b) KPIs-level validation - Modified Longest Common Subsequence

For the performance validation, we defined a KPI for the *average time between floors* as the time between the movement starting events (”Up,” ”Down”) and the stopping of the elevator (”Arrival”). We then used this metric for the alignment, and the results are displayed in Table IV. These results include a gradual increase in the ϵ value, which is the maximum difference between two values to consider them equivalent.

Based on these results, it can be concluded that 0.5 is the maximum difference between the furthest two values of the average time between floors. However, it is important to note that without considering the resulting sequence, it may be difficult to interpret which elements were not included and why. Furthermore, there may be cases where partial matches are found in certain regions of the sequence but are separated by a sequence of unaligned values. Unfortunately, such cases are not considered in this analysis, and the approach does not provide any assistance to the user in such situations.

TABLE IV: Performance-validation using LCSS statistics for scenario (4-3-2-1-0-1-2-3-4). Trace length of 1983 values of the average time between events.

ϵ	normalized score	score / LCSS length
0.1	0.099	197
0.2	0.099	197
0.3	0.224	444
0.4	0.722	1431
0.5	0.964	1911

c) KPIs-level validation - Dynamic Time Warping

For the analysis using Dynamic Time Warping, we use the same KPI presented in the previous section, the *average time between floors*. The output alignments using SbNDW and DTW, respectively, are available in Figure 5. In this figure, we can see how the value of the KPIs changes as the execution progresses. However, this alignment suffers heavily from over-stretching and

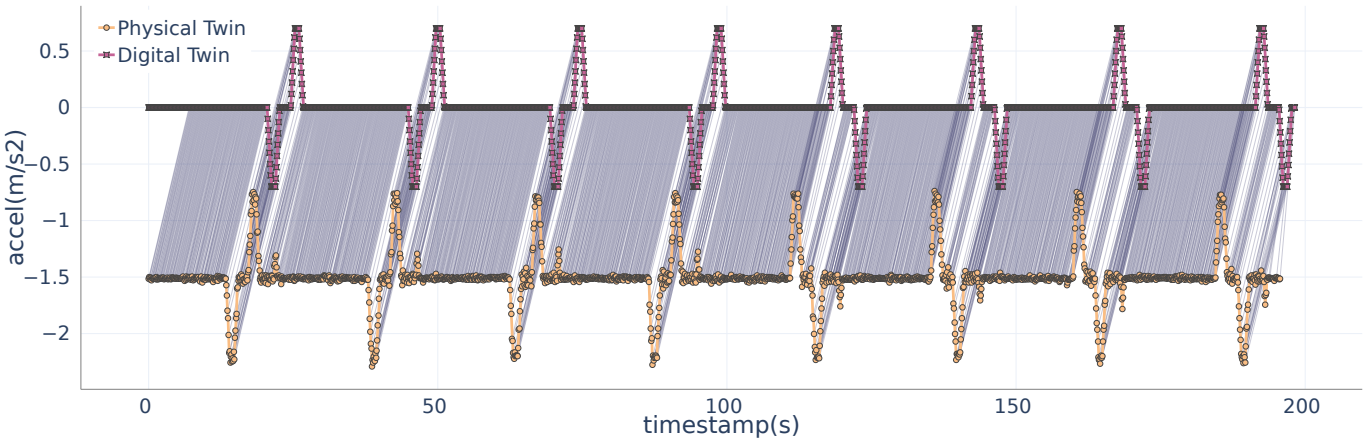
TABLE V: Dynamic Time Warping statistics for scenario (4-3-2-1-0-1-2-3-4)

Metric	DT	PT
Max times a snapshot is aligned	1783	1713
# Snapshots aligned more than once	2	2
Avg times a snapshot is aligned more than once	990	991
Std times a snapshot is aligned more than once	1121.47	1021.06
Total one-to-many alignments	1979	1978
Trace length	1983	
Score	27.67 seconds between events	
Normalized similarity score	0.98	

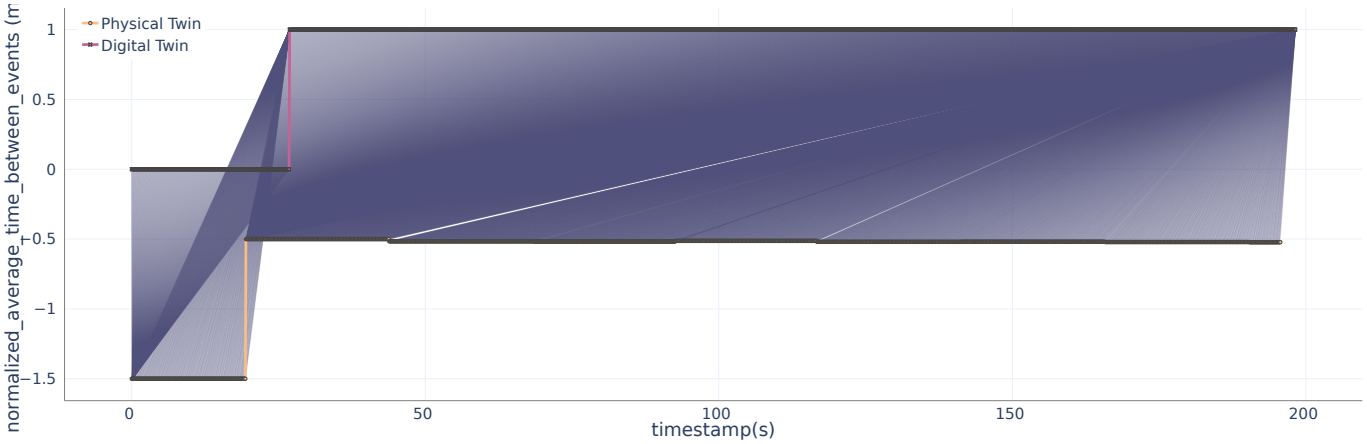
over-compressing, since a snapshot is aligned over 1700 times in each of the traces, as included in Table V. The results of an alignment so heavily influenced by this phenomenon may not be reliable for extracting solid conclusions.

Furthermore, it is challenging to detect subtle changes like small progressive delays based on an aggregated value such as the one shown in the graph, as these changes have minimal influence on the KPI. To address this issue, it is necessary to define performance indicators that are more sensitive to small changes in behavior, which would help in identifying and diagnosing problems.

The normalized similarity score proposed by the authors returns a value of 0.98, which is a similar value to the percentage of matched snapshots obtained in our proposal. However, due to the poor quality of the alignment, including over-stretching and over-over-compressing, and issues with normalization in the metric, doubts arise regarding the score's accuracy.



(a) LCAW (No Affine, LCAW)



(b) Lugaresi et al. approach with DTW

Fig. 5: Alignments for Scenario (4-3-2-1-0-1-2-3-4)

5 Performance analysis

Verifiability: The alignments performed in this section are available at https://github.com/atenearesearchgroup/fidelity-measure-for-dts/blob/main/src/resources/output/lift/performance

In this section, we will be comparing the performance of the NDW variants with the state-of-the-art algorithms we discussed earlier. To compare the algorithms, we ran alignments of the (4-3-2-1-0-1-2-3-4) scenario, gradually increasing the number of snapshots. If the scenario didn't include enough snapshots, we simulated a restart, running through it in a loop. We then measured the execution time for each alignment and attempted to perform a regression analysis to estimate the time complexity and compare it between the algorithms. Each alignment with a set length was repeated 5 times, and the average time and standard deviation were calculated.

5.1 Configuration

The performance analysis of algorithms should not be affected by the configuration used to execute them. However, to ensure reproducibility, we have provided the set configurations used for each algorithm that require parameters to produce a result. If an algorithm is not included in the following table, it means that it does not require any configuration parameter.

Parameter	**Value**
Needleman-Wunsch Variants	
Maximum Acceptable Distance (MAD)	0.15
Penalty opening a gap (P_{op})	-1
Penalty extending a gap (P_{ex})	-0.1
Low Complexity Areas Weight (LCAW)	0.005
LCSS Events	
Max time between matched snaps (δ)	6.0
LCSS KPIs	
Max distance between matched snaps (ϵ)	0.5

The algorithms were executed in the following computational environment:

CPU	Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz
RAM	64 GB
OS	Ubuntu (5.15.0-78-generic #85-Ubuntu SMP x86_64 GNU/Linux)

5.2 Results analysis

The performance results for each algorithm with different sequence lengths are available in Figure 6 and Table VI. To generate the graphics in Figure 6, we used polynomial regression analysis to determine the time complexity of each algorithm. The results of this analysis are available in Table VII.

As shown in Table VII, all of the algorithms exhibit approximately quadratic complexity, which is consistent with the theoretical complexity of dynamic programming algorithms. In these algorithms, one operation is performed for each position in the dynamic programming matrix. This matrix can have different height and width dimensions, but for simplicity, we analyze square matrices with the same height and length dimensions. We then consider a regression of the following form: $y = a \cdot x^b$, where x represents the sequence length, and y represents the time in seconds.

The values of 'a' are determined by the processor and the precise execution time of a snapshot. However, these values may vary depending on the specific processor used. Despite this, processing snapshots is relatively inexpensive, taking only around 100 nanoseconds on this processor. Comparing the values obtained for each algorithm, the higher the 'a' value, the steeper the slope, and the worse the algorithm's scalability.

In this case, the most expensive algorithm is the DTW version by Lugaressi. The reason for this is their implementation choice, as they decided to calculate the minimum value in each cell using the np.min function from numpy, which is more expensive

than the built-in min function in Python for small arrays. If they correct this issue, the execution time should be only slightly higher than the DTW algorithm we implemented for the comparison.

Following this DTW version by Lugaressi, we have the LCSS for Events, which has a similar performance. The algorithm LCSS with KPIs is the third least efficient, but it is slightly more efficient than the algorithm LCSS for events. This efficiency difference can be attributed to the string comparison, which is more expensive than the number comparison performed in the LCSS with KPIs. One solution to this problem is to replace the String identifiers with unique numeric values.

Let us focus on Subfigure 6b to analyze the remaining algorithms. The most efficient versions of NDW are the ones that do not include Affine Gap. This is because each cell's decision-making process only involves a matrix. On the other hand, Affine-Gap forces the algorithm to consult and modify three matrices in each step, which results in a 30.7% decrease in performance. LCAW also has additional execution time, but the difference is minimal. It is less than 1% in the case without Affine Gap, and 6% when Affine Gap is incorporated.

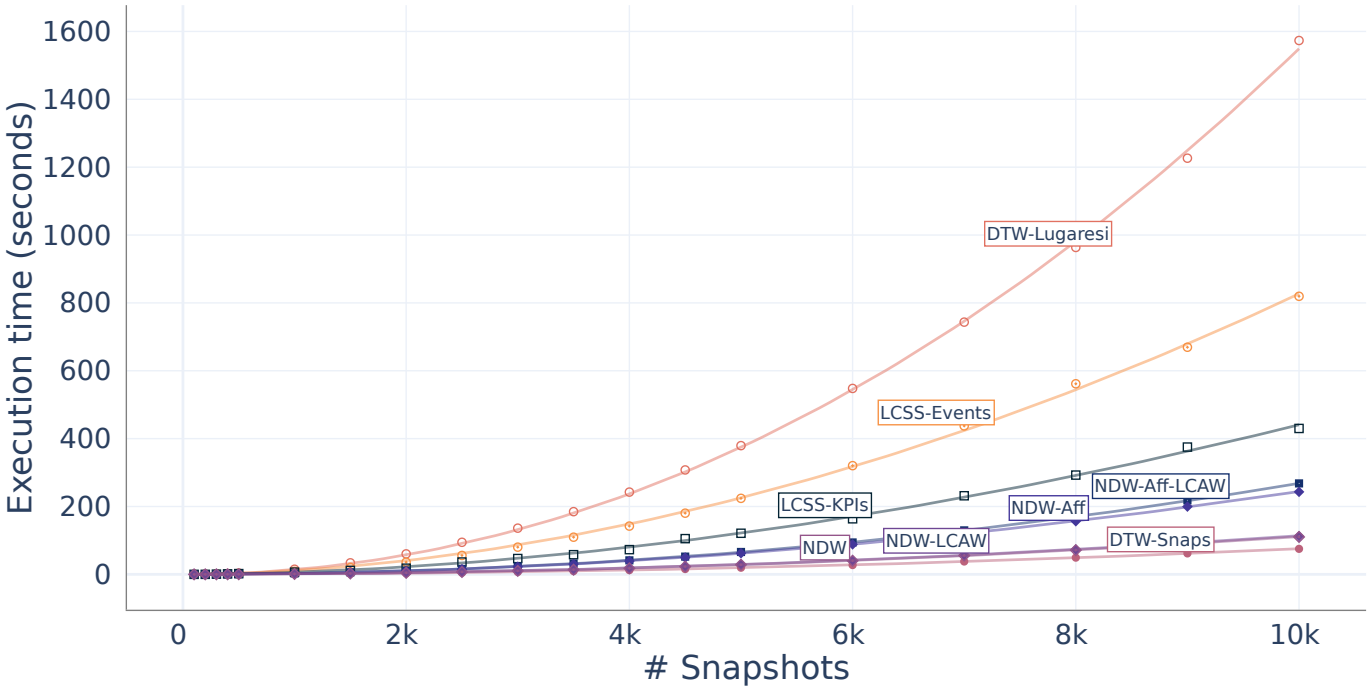
The DTW algorithm is the most efficient in terms of execution time (30% better than the base NDW variant), but produces lower-quality results compared to the base SbNDW, as explained in the previous section.

TABLE VI: Execution time for the alignment algorithms.

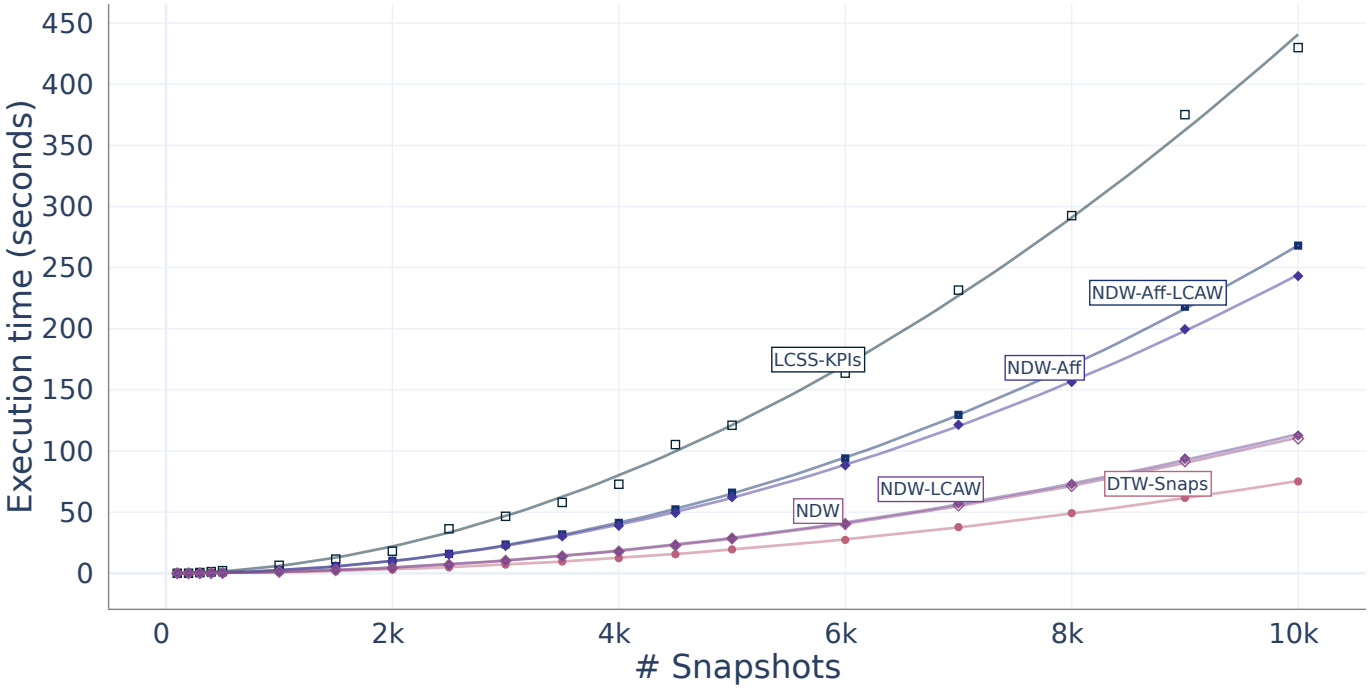
# snaps	NDW	NDW-LCAW	NDW-Aff	NDW-Aff-LCAW	DTW-Snaps	DTW-Lugaresi	LCSS-Events	LCSS-KPIs
100	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	0.15 ± 0.00	0.09 ± 0.09	0.01 ± 0.00
200	0.04 ± 0.00	0.04 ± 0.00	0.09 ± 0.00	0.10 ± 0.00	0.03 ± 0.00	0.60 ± 0.01	0.35 ± 0.37	0.06 ± 0.00
300	0.10 ± 0.00	0.10 ± 0.00	0.20 ± 0.00	0.21 ± 0.00	0.06 ± 0.00	1.34 ± 0.01	0.79 ± 0.83	0.62 ± 0.01
400	0.16 ± 0.00	0.16 ± 0.00	0.33 ± 0.00	0.35 ± 0.00	0.11 ± 0.00	2.40 ± 0.02	1.42 ± 1.50	1.37 ± 0.01
500	0.28 ± 0.00	0.28 ± 0.00	0.60 ± 0.01	0.62 ± 0.00	0.19 ± 0.00	3.81 ± 0.07	2.23 ± 2.35	2.17 ± 0.01
1000	1.19 ± 0.01	1.21 ± 0.01	2.60 ± 0.01	2.72 ± 0.02	0.81 ± 0.01	15.09 ± 0.12	8.79 ± 9.27	6.54 ± 0.04
1500	2.59 ± 0.02	2.62 ± 0.02	5.64 ± 0.03	5.92 ± 0.02	1.77 ± 0.02	33.95 ± 0.27	20.22 ± 21.31	11.61 ± 0.07
2000	4.43 ± 0.04	4.51 ± 0.03	9.68 ± 0.03	10.24 ± 0.06	3.06 ± 0.03	60.32 ± 0.43	35.86 ± 37.80	18.10 ± 0.16
2500	6.98 ± 0.05	7.04 ± 0.05	15.27 ± 0.13	15.94 ± 0.07	4.84 ± 0.04	94.49 ± 0.77	55.72 ± 58.72	36.40 ± 0.26
3000	10.25 ± 0.09	10.53 ± 0.08	22.39 ± 0.06	23.57 ± 0.08	7.13 ± 0.07	136.13 ± 1.62	80.21 ± 84.53	46.63 ± 0.26
3500	13.78 ± 0.03	14.19 ± 0.13	30.27 ± 0.13	31.84 ± 0.09	9.62 ± 0.08	184.77 ± 1.30	109.50 ± 115.40	57.96 ± 0.33
4000	17.82 ± 0.07	18.16 ± 0.19	38.89 ± 0.30	41.24 ± 0.18	12.35 ± 0.04	242.43 ± 0.49	142.27 ± 149.93	72.84 ± 0.36
4500	22.79 ± 0.21	23.29 ± 0.21	49.40 ± 0.23	52.29 ± 0.35	15.57 ± 0.12	307.70 ± 2.79	180.30 ± 190.01	105.27 ± 0.62
5000	28.45 ± 0.23	28.88 ± 0.12	62.25 ± 0.31	65.98 ± 0.11	19.53 ± 0.12	379.34 ± 3.11	224.03 ± 236.09	121.08 ± 0.36
6000	40.23 ± 0.22	40.86 ± 0.23	88.32 ± 0.48	93.98 ± 0.55	27.31 ± 0.27	547.94 ± 5.47	320.32 ± 337.57	163.72 ± 0.72
7000	55.48 ± 0.27	57.30 ± 0.30	121.48 ± 0.33	129.60 ± 0.66	37.80 ± 0.41	743.34 ± 6.89	437.43 ± 461.10	231.62 ± 0.98
8000	71.38 ± 0.15	73.12 ± 0.31	156.51 ± 1.22	167.91 ± 0.40	49.24 ± 0.72	963.42 ± 8.18	561.69 ± 591.94	292.53 ± 1.44
9000	91.56 ± 0.23	93.91 ± 0.31	199.60 ± 1.12	218.06 ± 3.92	61.71 ± 0.60	1226.47 ± 10.15	669.34 ± 705.52	375.12 ± 1.46
10000	110.16 ± 2.12	112.75 ± 1.00	243.10 ± 1.86	268.03 ± 19.51	75.11 ± 0.83	1573.30 ± 81.39	819.49 ± 863.60	429.95 ± 2.65

TABLE VII: Regression analysis results for the alignment algorithms

Algorithm name	Regression result	Time Complexity
NDW-Aff-LCAW	$y = 1.83 \cdot 10^{-6} x^{2.04}$	$O(n^{2.04}) \approx O(n^2)$
NDW-Aff	$y = 2.82 \cdot 10^{-6} x^{1.98}$	$O(n^{1.98}) \approx O(n^2)$
NDW-LCAW	$y = 1.40 \cdot 10^{-6} x^{1.98}$	$O(n^{1.98}) \approx O(n^2)$
NDW	$y = 1.40 \cdot 10^{-6} x^{1.97}$	$O(n^{1.97}) \approx O(n^2)$
DTW-Snaps	$y = 1.07 \cdot 10^{-6} x^{1.96}$	$O(n^{1.96}) \approx O(n^2)$
DTW-Lugaresi	$y = 9.92 \cdot 10^{-6} x^{2.05}$	$O(n^{2.05}) \approx O(n^2)$
LCSS-Events	$y = 26.0 \cdot 10^{-6} x^{1.88}$	$O(n^{1.88}) \approx O(n^2)$
LCSS-KPIs	$y = 15.7 \cdot 10^{-6} x^{1.86}$	$O(n^{1.86}) \approx O(n^2)$



(a) Execution time comparison for all algorithms.



(b) Execution time comparison for all algorithms except Lugaresi's adaption of DTW.

Fig. 6: Execution time by the number of snapshots in Scenario (4-3-2-1-0-1-2-3-4)

References

[1] G. Lugaresi, S. Gangemi, G. Gazzoni, and A. Matta, "Online validation of digital twins for manufacturing systems," *Comput. Ind.*, vol. 150, p. 103942, 2023.

[2] 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.

[3] 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/

[4] *Sequence Alignment (I)*. John Wiley & Sons, Ltd, 2021, ch. 3, pp. 51–97.

[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. 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.

[7] D. Mount, *Bioinformatics: Sequence and Genome Analysis*, ser. Cold Spring Harbor Laboratory Series. Cold Spring Harbor Laboratory Press, 2004. [Online]. Available: https://books.google.es/books?id=bvY21DGa1OwC

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

[9] R. Bellman, *Dynamic Programming*. Dover Publications, 1957.

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

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

[12] 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. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.89.22.10915

[13] Peters Research, "Elevate software," 2023. [Online]. Available: https://peters-research.com/index.php/elevate/

[14] P. Muñoz, J. Troya, M. Wimmer, and A. Vallecillo, "Using trace alignments for measuring the fidelity of a physical and a digital twin: Elevator technical report," 2023. [Online]. Available: https://github.com/atenearesearchgroup/fidelity-measure-for-dts/blob/main/docs/Technical_Report_Elevator.pdf

[15] ——, "Using trace alignments for measuring the fidelity of a physical and a digital twin: General concepts," 2023. [Online]. Available: https://github.com/atenearesearchgroup/fidelity-measure-for-dts/blob/main/docs/Technical_Report_GeneralConcepts.pdf

[16] H. Sakoe and S. Chiba, "A dynamic programming approach to continuous speech recognition," in *Proc. of the 7th International Congress on Acoustics*, vol. 3. Akadémiai Kiadó, 1971, pp. 65–69.

[17] H. Li, J. Liu, Z. Yang, R. W. Liu, K. Wu, and Y. Wan, "Adaptively constrained dynamic time warping for time series classification and clustering," *Information Sciences*, vol. 534, pp. 97–116, 2020.