Generalizing identity-based string comparison metrics: Framework and Techniques

DOI: https://doi.org/10.1016/j.knosys.2019.06.028

Generalizing identity-based string comparison metrics: Framework and Techniques

Francesco Cauteruccio[1], Giorgio Terracina[1], and Domenico Ursino[2]
[1] DEMACS, University of Calabria,
[2]DII, Polytechnic University of Marche
{cauteruccio,terracina}@mat.unical.it, d.ursino@univpm.it

Abstract

In this paper, we propose a framework that aims at handling metrics among strings defined over heterogeneous alphabets. Furthermore, we illustrate in detail its application to generalize one of the most important string metrics, namely the edit distance. This last activity leads us to define the Multi-Parameterized Edit Distance (MPED). As for this last metric, we investigate its computational properties and solution algorithms, and we present several experiments for its evaluation. As a final contribution, we provide several notes about some possible applications of MPED and other generalized metrics in different scenarios.

Keywords: String Metrics, Generalized String Similarity Framework, Matching Schema, Generalized Metric Function, Multi-Parameterized Edit Distance

1 Introduction

Ordered sequences of symbols (also called strings) play an important role in computer science. Indeed, with the support of an adequate semantics, they can be used to express several kinds of data. Data provided as streams of strings are constantly increasing; think, for instance, of sensor networks, wearable devices, distributed agents, sequences of events, etc. Interactions among such data streams, and consequently among the corresponding string representations, arise some intriguing questions, like *How much two strings are correlated?*, or *How (dis)similar are they?*.

Several techniques for string comparison have been proposed in past literature. Basically, a measure of (dis)similarity between strings is a function that takes two strings s_1 and s_2 as input and returns a value d, representing the level of (dis)similarity of s_1 and s_2 based on some metric.

String metrics may significantly differ for the rules adopted to measure the (dis)similarity degree; in their turn, these rules depend on the context in which they are applied [17]. However, most of the available metrics are based on the natural assumption that identical symbols among strings represent identical information, whereas different symbols introduce, in a way or another, some form of differentiation.

This assumption, even if generally valid, could fail in some circumstances. For instance, consider two strings $s_1 =$ `AAABCD` and $s_2 =$ `111234`; any standard metric would state that they are completely

different. Nevertheless, there are cases (like the one shown above) in which symbol identity seems to be not enough. In fact, even if there are no common symbols between two strings, it could happen that they represent similar information to some extent.

What happens if we strongly believe that there is some underlying matching between two strings that apparently have different symbols but similar structures? As an example, consider two strings s_1 and s_2 and assume that they are generated by heterogeneous data streams, derived from two different ways of measuring the same reality; think, for instance, of two sensors one measuring light and the other measuring temperature. The values, the scales and the meaning of the two sensors may be very different but, if sensors are near to a fire, light can be influenced by temperature, and vice versa. To properly monitor this event, we should be able to understand the correlation between these two heterogeneous measurements. As a further example, suppose that s_1 and s_2 have been *deliberately* manipulated in such a way as to appear dissimilar (for instance, using code cloning techniques) [30, 43, 24], even if their meaning is actually identical.

In these cases, the necessity arises of a suitable metric capable of capturing hidden correlations between strings. This metric should take into account that *different* symbols in the involved strings may express *similar concepts*.

A step forward in this context, in particular for code cloning detection, has been carried out with the introduction of *parameterized strings*, i.e., strings which have some symbols acting as parameters that can be substituted at no cost. In this setting, a seminal approach has been presented in [6], and some variants and extensions of the metric introduced therein have been proposed in past literature [4, 7, 28, 31]. Among these extensions, the ones using injective and bijective mapping functions are extremely important in our context. In fact, there are many application scenarios where one-to-one mappings do not sufficiently express concept similarities among heterogeneous strings (see Section 2 for more details). In spite of the interesting ideas underlying all the approaches mentioned above, they suffer from an important limitation in that they are tailored to specific problems (e.g., code cloning detection) or to specific metrics.

This paper aims at taking a further step forward in this direction. Indeed, it presents a framework that generalizes most of the existing string metrics, making them suitable for application scenarios where involved strings could be based on heterogeneous alphabets. In this way, our approach makes the adoption of string metrics possible in all those contexts in which involved strings come from different sources, each using its own alphabet.

The main components of the proposed framework are: *(i)* a *matching schema*, which formalizes matches between symbols, and *(ii)* a *generalized metric function*, which abstracts the computation of string metrics based on a pre-defined matching schema. The proposed framework is based on the identification of the best matching schema for the metric function, i.e., the matching schema leading to the minimum value of the distance function when applied to the strings into consideration.

To better illustrate the behavior and the contribution of our framework, we describe its application to the generalization of one of the most important string metrics, namely edit distance. As for this issue, we provide both a theoretical study of the computational costs related to such generalization and solution techniques for its implementation.

Just to provide an example of the potential of our approach, given the strings $s_1 =$ `AAABCCDDCAA` and $s_2 =$ `EEFGHGGFHH`, any classical method would state that they are completely different. By

contrast, our generalization of the edit distance would find that, by matching the pair of symbols {A,B} with the pair {E,H} and the pair {C,D} with the pair {F,G}, the following alignment is possible:

$$
\begin{aligned}
s_1 &: \texttt{AAABCCDDCAA} \rightarrow \texttt{AAABCCDDCAA} \\
s_2 &: \texttt{EEFGHGGFHH} \ \rightarrow \texttt{-EEFGHGGFHH} \\
&\qquad\qquad\qquad\qquad\quad \texttt{** * *****}
\end{aligned}
$$

which states that the second string could be obtained from the first one with just 3 (parameterized) edit operations, thus finding a significant, not obvious, correlation between the two strings (see Section 4 for all details).

A preliminary version of the generalized edit distance has been formerly introduced in [11]. In this paper, we significantly extend that work by providing the following new contributions:

- We provide a formalization of the framework.
- We formally prove that it generalizes most of the existing string metrics, making them suitable for application scenarios where involved strings could be based on heterogeneous alphabets.
- As for the generalization of the edit distance (which we call Multi-Parameterized Edit Distance - MPED), we provide a detailed description of its theoretical implications; in particular, we prove that its computation is NP-Hard.
- Based on the previous result, we introduce the definition of an efficiently computable lower bound for MPED.
- We investigate several implementations of MPED, based on heuristic approaches.
- We extensively test all the presented implementations, we compare their performances and we investigate the strengths and the weaknesses of each of them.
- We describe three application scenarios that can benefit from MPED.

The rest of this paper is organized as follows. Section 2 reviews related literature. Section 3 introduces the proposed framework and an overview of its application to generalize some of the most important metrics for string comparison already proposed in the past. Section 4 is devoted to study in all detail MPED, which represents the generalization of the edit distance obtained by applying the proposed framework. In the same section, we also provide the computational properties and some solution algorithms for MPED. The experimental evaluation of MPED is discussed in Section 5. Section 6 provides some notes about three possible applications of the proposed generalized metric. Finally, in Section 7, we draw our conclusions and highlight some future directions of our research efforts.

2 Related work

String similarity computation has been a challenging issue in past literature, and several attempts to face this problem have been presented. However, most of the literature generalizing classical approaches with parameterized alphabets focuses on the pattern matching problem [6], whose objective is to seek (exact or approximate) occurrences of a given pattern in a text. This is the context in which parameterized strings were introduced. Here, some of the symbols act as parameters that can be properly substituted at no cost.

A seminal work on this topic is presented in [6]. The approach described in this paper compares parameterized strings. It considers *bijective global* transformation functions allowing *exact p-matches* only. This means that the two strings could match only if they have the same length; thus, no substitutions or insertions are possible.

Mismatches are allowed in [28], where the authors face the problem of finding all the locations in a string s in which there exists a *global bijection* π that maps a pattern p into the appropriate substring of s minimizing the Hamming distance. In this case, *injective* functions, instead of bijective ones, are considered in [4].

String matching has been extensively used for clone detection, i.e., to check if a code contains two or more cloned parts. In [30], a token-based code clone detection approach is presented. Here, matches between strings are carried out by using a suffix-tree algorithm. The code is tokenized by one parameter only; thus, a match is represented by a one-to-many mapping.

In [3], the authors introduce the concept of *generalized function matching* applied to the pattern matching problem in several contexts, like image searching, DNA analysis, poetry and music analysis, etc.

A detailed survey on parameterized matching appears in [37].

Moving towards string similarity computation, a relevant research issue regards the longest common subsequence problem (hereafter, LCS) and its parameterized versions. In particular, in [31], the parameterized version of the LCS problem is considered. Interestingly, LCS allows only insertions and deletions, but no substitutions.

String similarity metrics present important overlap with approximate pattern matching, since one can determine the distance between two strings by asking whether there exists an approximate pattern matching with at most k mismatches. However, having a direct approach for measuring the parameterized distance provides obvious benefits.

Few works consider the problem of parameterized distances between strings. In [7], the notion of *p-edit* distance is introduced. It focuses on the edit distance, where allowed edit operations are *insertions*, *deletions* and *exact p-matches*. Mismatches are not allowed. Furthermore, two substrings that participate in two distinct exact p-matches are independent of each other, so that mappings have local validity over substrings not broken by insertions and deletions. In particular, within each of these substrings, the associated mapping function must be bijective. The work presented in [31] extends the approach proposed in [7] by requiring that the transformation function has a global validity; however, it still limits the set of allowed edit operations (in particular, substitutions are not allowed).

The work in [22] is based on the approach proposed in [6]; it introduces an order-preserving match, but it limits the number of mismatches to k. In [27], a preliminary approach to a *many-*

to-many mapping function for string alignment can be found; it computes alignments between two parameterized strings and gives preference to alignments based on the co-occurrence frequency.

Finally, there are a high number of applications in which string similarity computation plays a key role. In [41], a biology-inspired data mining framework for extracting patterns in sexual cyberbullying data is defined; this topic has received much attention in the latest years. This framework is based on the multiple sequence alignment method and aims at recognizing bullying patterns within the questions posed by a predator to his victims. In [48], instead, a pattern recognition system that detects malware on a mobile operating system, based on the analysis of suspicious boot sequences, is developed. This system uses sequence alignment for assessing similarity between legitimate and malicious system call sequences, in such a way as to discriminate sequences belonging to a malware from the ones concerning a normal process. Different techniques used in the context of time series similarity analysis stem from the string similarity ones. For example, in [44], an empirical evaluation of similarity measures for time series classification is presented. This approach leverages measures such as dynamic time warping and edit distance for real sequences.

3 A framework for handling generalized string metrics

As we pointed out in the Introduction, the key components of the proposed framework are matching schemas, which generalize symbol identity to symbol match, and generalized metric functions, which substitute symbol identity with the symbol matches allowed by the matching schemas at hand. Given two input strings s_1 and s_2, a set $\mathbb{M}$ of *valid* matching schemas (see below) and a generalized metric function $f^M(\cdot,\cdot)$, our framework aims at computing the minimum value for $f^M(s_1, s_2)$ which can be obtained by any matching schema M in $\mathbb{M}$.

Our framework can be formally defined by a tuple:

$$\mathfrak{F} = \langle \Pi_1, \Pi_2, \pi_1, \pi_2, \chi, f^M(\cdot,\cdot)\rangle$$

where:

- Π_1 and Π_2 are the alphabets on which the strings under consideration are defined.
- π_1, π_2 and χ are parameters that determine the shape of all valid matching schemas.
- $f^M(\cdot,\cdot)$ is a *generalized metric function*.

In particular, for each pair of symbols in Π_1 and Π_2, a matching schema M states whether they match or not. π_1 (resp., π_2) determines how many symbols of Π_1 (resp., Π_2) can have a symbol match with one symbol of Π_2 (resp., Π_1). Conversely, χ states the pairs of symbols from Π_1 and Π_2 that can never have a symbol match in a valid matching schema. Valid matching schemas are also not ambiguous in determining the mutual matchings between the symbols in Π_1 and Π_2; in particular, if $a \in \Pi_1$ matches with $b \in \Pi_2$, then also b matches with a. Given Π_1, Π_2, π_1, π_2, and χ, the set of all valid matching schemas is represented by $\mathbb{M}$.

An intuitive example of a valid matching schema is graphically depicted in Figure 1 for $\Pi_1 = \{a1, a2, a3,\ a4, a5, a6, a7, a8, a9, a10\}$, $\Pi_2 = \{b1, b2, b3, b4, b5, b6, b7, b8, b9\}$, $\pi_1 = 3$ and $\pi_2 = 2$. Black

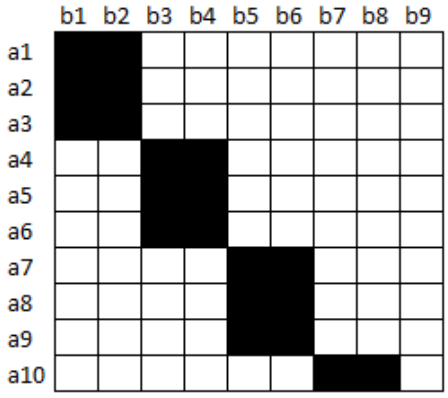

Figure 1: Example of a matching schema for $\pi_1 = 3$ and $\pi_2 = 2$

cells indicate matches, whereas white cells represent mismatches. In particular, in this example, $a1$ matches with both $b1$ and $b2$; analogously, $b2$ matches with both $a1$, $a2$, and $a3$; more generally, the set of symbols $\{a1, a2, a3\}$ match with the set of symbols $\{b1, b2\}$. Observe that, in this case, matching schemas fully generalize the concept of symbol identity, since the two alphabets are completely disjoint. The formal definition of the concept of matching schema can be found in [11].

When applied on two strings s_1 and s_2 over the alphabets Π_1 and Π_2, respectively, $\mathfrak{F}$ returns the minimum value of $f^M(s_1, s_2)$ that can be obtained by taking any matching schema M of $\mathbb{M}$.

Observe that, in the construction of $\mathfrak{F}$, having defined Π_1, Π_2, π_1, π_2, and χ implies that also the set $\mathbb{M}$ of all the valid matching schemas is implicitly defined.

In what follows, the length of a string s_i ($i \in \{1, 2\}$), i.e., the number of its symbols, will be denoted by $len(s_i)$. Moreover, for each position $1 \leq j \leq len(s_i)$, the j-th symbol of s_i will be identified by $s_i[j]$ and the substring of s_i starting at position x and ending at position y will be denoted as $s_i[x..y]$.

3.1 Generalization of notable string similarity metrics

In the previous section, we illustrated our framework $\mathfrak{F}$, which paves the way to quite a general computation of string (dis)similarity.

In this section, we show that $\mathfrak{F}$ is general enough to encompass several classical and notable string similarity metrics. First of all, we show that a particular specialization of $\mathfrak{F}$ includes both character-based and token-based distances; then, we show that $\mathfrak{F}$ can be also specialized to more sophisticated comparison approaches, like the *parameterized pattern matching* proposed in [6].

Proposition 3.1 Given the following specialization of $\mathfrak{F}$:

1. $\pi_1 = \pi_2 = 1$;
2. $\chi = \{(c_i, c_j) | c_i \in \Pi_1, c_j \in \Pi_2, c_i \neq c_j\}$;

then, $f^M(\cdot, \cdot) = f(\cdot, \cdot)$ for all the metric functions $f(\cdot, \cdot)$ based on symbol identity. □

Intuitively, with $\pi_1 = \pi_2 = 1$, the cardinality of each matching subset is equal to 1, and this constraints to one-to-one symbol matchings. With the provided construction of χ, only symbol identities are allowed. As a consequence, only one matching schema M is valid, namely the one stating that, for each pair of symbols $\alpha \in \Pi_1$ and $\beta \in \Pi_2$, there is a match between α and β if and only if $\alpha = \beta$.

This matching simulates symbol identity specification for the generalized metric functions based on symbol identity.

Edit Distance In its simplest form, the edit distance between two strings s_1 and s_2 is the minimum number of edit operations (insertions, deletions or substitutions) performed on single characters needed to transform s_1 into s_2, where each operation has a cost equal to 1 [34, 40]. This version of the edit distance is also referred to as Levenshtein distance. Some variants of the edit distance allow for arbitrary edit costs; these versions are particularly useful, for instance, in bioinformatics.

A basic algorithm for edit distance computation exploits a dynamic programming approach; in it, the choice of the edit operation is carried out by a recurrence formula, where the discriminating factor is based on the question "Is $s_1[i] = s_2[j]$?". If our framework $\mathfrak{F}$ is applied, this question is substituted by the following one: "According to M, do the symbols $s_1[i]$ and $s_2[j]$ match?". As specified by Proposition 3.1, according to the specialization defined therein, this question is equivalent to asking for symbol identity.

It is worth pointing out that edit distance should not be confused with sequence alignment scores based on substitution matrices (e.g., the alignment algorithms using PAM or BLOSUM substitution matrices). In fact, even if these approaches are based on a similar dynamic programming backbone, they rely on a substitution coefficient defined for every pair of symbols. As a consequence, the basic question is no longer "Is $s_1[i] = s_2[j]$?", but rather "how much would it cost to substitute $s_1[i]$ with $s_2[j]$?". Actually, in this case, our framework cannot be applied.

Affine Gap Distance The affine gap distance metric [50] is similar to the edit distance, except for the fact that it introduces two extra edit operations, i.e., *open gap* and *extend gap*. In this way, the cost of the first insertion (gap opening) can be different from, and is usually higher than, the one of adding consecutive insertions (gap extensions). Similarly to the edit distance, the discriminating factor in computing the affine gap distance is based on the question "Is $s_1[i] = s_2[j]$?". When applying our framework $\mathfrak{F}$ with the specialization introduced in Proposition 3.1, this question is substituted by the one "According to M, do the symbols $s_1[i]$ and $s_2[j]$ match?", which has been shown to be equivalent to asking for symbol identity.

Smith-Waterman Distance The Smith-Waterman distance [46] is an extension of both the edit distance and the affine gap distance, in which mismatches at the beginning and at the end of strings have lower costs than mismatches in the middle. This metric allows for substring matchings and is well suited for fitting shorter strings into longer ones. Also in this case, the basic resolution schema exploits dynamic programming, where the discriminating factor in the recurrence formula is the question "Is $s_1[i] = s_2[j]$?".

Jaro Distance Metric The Jaro distance metric, introduced in [29], is based on the concepts of common characters and transpositions. Given two strings s_1 and s_2, two symbols $s_1[i]$ and $s_2[j]$ are *common characters* when $s_1[i] = s_2[j]$ and $|i - j| \leq \frac{1}{2}\min\{len(s_1), len(s_2)\}$. Given the i-th common character a in s_1 and the i-th common character b in s_2, if $a \neq b$ this is a transposition.

The Jaro distance value is, then, computed as:

$$\text{Jaro}(s_1, s_2) = \frac{1}{3}\left(\frac{c}{len(s_1)} + \frac{c}{len(s_2)} + \frac{c - \frac{1}{2}t}{c}\right)$$

where c is the number of common characters and t is the number of transpositions.

The characteristics of the Jaro distance clearly differ from the ones of the aforementioned metrics. However, as a common core for the definitions of common character and transposition, there is, again, symbol equality, where the discriminating question is "Is $s_1[i] = s_2[j]$?".

Also in this case, it is easy to show that substituting this question with the one "According to M, do the symbols $s_1[i]$ and $s_2[j]$ match?", under the specialization of Proposition 3.1, makes the two settings equivalent.

Atomic Strings In atomic strings, the comparison shifts away from single character to longer strings. In particular, s_1 and s_2 are tokenized by punctuation characters. Each token is called *atomic string*. Two atomic strings match if they are equal or if one is a prefix of the other. The similarity between s_1 and s_2 is, then, computed as the fraction of the atomic strings that match.

Observe that, even if this metrics moves from the comparison of single characters to the one of substrings, the basic operation used to identify a matching for atomic strings is the identity of sequences of single characters. As a consequence, the same considerations outlined above when substituting the question "Is $s_1[i] = s_2[j]$?" with the one "According to M, do the symbols $s_1[i]$ and $s_2[j]$ match?" are still valid.

WHIRL In [15], the cosine similarity is combined with the *tf.idf* weighting scheme to obtain the WHIRL system aiming at comparing pairs of strings in a set of records. In particular, each string s is separated into words; a weight $v_s(w)$ is assigned to each word w of s; $v_s(w)$ depends on the number tf_w of times when w appears in s and on the fraction idf_w of records containing w. The cosine similarity between two strings s_1 and s_2 is, then, defined as:

$$\text{sim}(s_1, s_2) = \frac{\Sigma_w v_{s_1}(w) \cdot v_{s_2}(w)}{||v_{s_1}||^2 \cdot ||v_{s_2}||^2}$$

Despite the complexity of this metric, as for its application to our context, the most relevant thing to observe is that both the tf_w and the idf_w components are based on the exact occurrence of w in the string(s). As a consequence, again, when asking whether a word w is contained into a string s, the basic question for the computation is "Is $w[i] = s[j]$?", which, as previously shown, can be simulated by the question "According to M, do the symbols $w[i]$ and $s[j]$ match?", under the specialization of Proposition 3.1.

Q-grams with *tf.idf* Q-grams with *tf.idf* [26] extends the metric adopted in WHIRL by using q-grams, instead of words. This allows the management of spelling errors, as well as the insertion and the deletion of words. The computation setting is equal to the one of WHIRL; therefore, all the considerations about the specialization of $\mathfrak{F}$ seen for WHIRL can be applied also to this metrics.

Parameterized pattern matching In [6], an approach to compare strings over partially overlapping alphabets is proposed. This approach, called *parameterized pattern matching*, aims at identifying pairs of strings being equal except for a one-to-one symbol substitution. In particular, the alphabet of each string s_i is partitioned in two alphabets, namely Σ_i and Π_i; the former contains standard symbols and the latter encompasses parameters. These last can be renamed at no cost. Two such strings identify a parameterized match if one of them can be obtained by renaming the parameters of the other by means of a one-to-one function. This approach has been shown to be particularly useful in code cloning identification.

Our framework can be specialized to accommodate parameterized pattern matching. In fact, given two strings s_1 and s_2, defined over the alphabets $\Sigma_1 \cup \Pi_1$ and $\Sigma_2 \cup \Pi_2$, respectively, consider the following specialization of $\mathfrak{F}$:

1. $\pi_1 = \pi_2 = 1$;
2. $\chi = \{(c_i, c_j) | c_i \in \Sigma_1, c_j \in \Sigma_2, c_i \neq c_j\}$[1]

Let $f(\cdot, \cdot)$ be the Hamming distance (which, basically, is the edit distance allowing symbol substitutions only). If the minimum $f^M(s_1, s_2) = 0$, then there is a parameterized matching between s_1 and s_2.

In particular, $\pi_1 = \pi_2 = 1$ limits to one-to-one functions. The definition of χ states that the only valid match configuration between pairs of symbols in Σ_1 and Σ_2 is symbol identity, whereas any symbol in Π_1 (resp., in Π_2) can be matched at no cost with any symbol of the other string by means of a one-to-one matching function. $\mathfrak{F}$ finds the minimum value of the Hamming distance, among all the possible one-to-one substitutions. A value of this distance equal to 0 denotes that one string can be obtained by renaming the parameters of the other by means of a one-to-one function and, consequently, that a parameterized match holds.

4 Customizing our framework to edit distance

In the previous section, we introduced our framework $\mathfrak{F}$ for generalizing string metrics. We also showed that $\mathfrak{F}$ is general enough to accommodate the classical definitions of well known string metrics. In this section, we show how it can also be applied to generalize a classic metric, namely the edit distance, in such a way that symbol identity is substituted by many-to-many symbol correlations, where identifying the best matching schema is part of the problem. We call MPED (Multi-Parameterized Edit Distance) the new generalized metric. A preliminary formal definition of MPED has been previously introduced in [11]; here, we provide it as a specialization of our framework. Furthermore, we provide a detailed

[1] Observe that, while in Proposition 3.1 we consider the whole alphabet, here we are specifying only Σ_1 and Σ_2.

description of the theoretical implications of MPED by focusing on its computational complexity, and we introduce the definition of an efficiently computable lower bound. Finally, we introduce different implementations of MPED, along with a thorough experimental evaluation comparing their performances.

If we specialize our framework $\mathfrak{F}$ to MPED we obtain:

$$\mathfrak{F}_{\mathcal{L}} = \langle \Pi_1, \Pi_2, \pi_1, \pi_2, \chi, \mathcal{L}^M(\cdot,\cdot) \rangle$$

$\mathcal{L}^M(\cdot,\cdot)$ is the generalized version of $\mathcal{L}(\cdot,\cdot)$ that takes the parameters π_1, π_2, and χ into account. $\mathcal{L}(\cdot,\cdot)$ is the classical edit distance.

In the following, we use $\mathcal{L}_{\langle \pi_1,\pi_2,\chi \rangle}(s_1, s_2)$ to denote the minimum value of $\mathcal{L}^M(s_1, s_2)$ computed for $\mathfrak{F}_{\mathcal{L}}$ among all valid matching schemas, according to π_1, π_2, and χ.

Recall that the edit distance is based on the concept of minimum cost alignment of two strings, where edit operations are substitutions, insertions, and deletions (also known as indels), and each edit operation has a cost. In our context, a substitution occurs when two symbols do not match, according to M. The interested reader is referred to [34, 40] for all the details about the classical methods for the computation of the edit distance, which, due to space constraints, are not reported here.

The exact value of $\mathcal{L}_{\langle \pi_1,\pi_2,\chi \rangle}(s_1, s_2)$ may be computed by means of a brute force approach, which we call `EX` in the following. This approach explores all the possible valid matching schemas and determines the one that returns the lowest edit distance. Obviously, `EX` is unfeasible for large values of $|\Pi_1|$, $|\Pi_2|$, π_1 and π_2.

Example 4.1 Let s_1 = `AAABCCDDCAA` and s_2 = `EEFGHGGFHH`, and let Π_1 = {`A,B,C,D`} and Π_2 = {`E,F,G,H`}. For $\pi_1 = \pi_2 = 1$, the best alignment $\langle \bar{s}_1, \bar{s}_2 \rangle$, which can be computed by `EX`, is obtained by matching {`A`} with {`E`}, {`B`} with {`G`}, {`C`} with {`H`}, and {`D`} with {`F`}. This alignment is[2]:

```
s1 : AAABCCDDCAA → AAABCCDDCAA
s2 : EEFGHGGFHH  → EEFGHGGFH-H
                   ** **  **
```

It returns $\mathcal{L}_{\langle \pi_1,\pi_2,\chi \rangle}(s_1, s_2) = 5$. Observe that this approach works properly even if $\Pi_1 \cap \Pi_2 = \emptyset$ and the input strings have different lengths.

If we set $\pi_1 = \pi_2 = 2$, the best alignment, which can be computed by `EX`, is the one obtained by matching {`A,B`} with {`E,H`}, and {`C,D`} with {`F,G`}, namely:

```
s1 : AAABCCDDCAA → AAABCCDDCAA
s2 : EEFGHGGFHH  → -EEFGHGGFHH
                    ** * *****
```

It returns $\mathcal{L}_{\langle \pi_1,\pi_2,\chi \rangle}(s_1, s_2) = 3$.

[2]Here and in the following, we use the standard notation for representing string alignments (see [34, 40]).

Suppose, now, to introduce the constraint $\chi = \{(\texttt{A}, \texttt{E})\}$. For $\pi_1 = \pi_2 = 1$, the best alignment $\langle \bar{s}_1, \bar{s}_2 \rangle$ computed by `EX` is the following:

$$
\begin{aligned}
s_1 &: \texttt{AAABCCDDCAA} \rightarrow \texttt{AAABC-CDDCAA} \\
s_2 &: \texttt{EEFGHGGFHH} \ \rightarrow \texttt{--EEFGHGGFHH} \\
&\qquad\qquad\qquad\qquad\quad \texttt{**} \ \ \texttt{*****}
\end{aligned}
$$

It still returns $\mathcal{L}_{\langle \pi_1, \pi_2, \chi \rangle}(s_1, s_2) = 5$; however, in this case, `A` and `E` do not match anymore. Optimal matchings are, in fact, {`A`} with {`H`}, {`B`} with {`E`}, {`C`} with {`F`}, and {`D`} with {`G`}. □

4.1 Computational issues

After having defined $\mathfrak{F}_{\mathcal{L}}$, it is important to analyze its properties. We start by determining its computational complexity. The specialization of $\mathfrak{F}$ provided in Proposition 3.1 ensures that, in that case, there exists only one valid matching schema in $\mathbb{M}$, and this can be easily derived by construction. However, in general, $\mathbb{M}$ could contain several matching schemas; each of them could match π_1 symbols of Π_1 with π_2 symbols of Π_2; furthermore, for each matching schema, several alignments between s_1 and s_2 are possible.

In the following, we show that the general problem of computing MPED is NP-Hard.

Theorem 4.1 The problem of computing $\mathcal{L}_{\langle \pi_1, \pi_2, \chi \rangle}(s_1, s_2)$ is NP-Hard.

Proof.

This theorem can be proven by performing a reduction from the three-dimensional matching problem (hereafter, 3DM) [20]. First recall that 3DM is defined as follows:

> Let X, Y and Z be finite and disjoint sets, and let M be a subset of $X \times Y \times Z$. Therefore, M consists of triples $\langle X_i, Y_i, Z_i \rangle$ such that $X_i \in X$, $Y_i \in Y$, and $Z_i \in Z$. Now, $M' \subseteq M$ is a 3DM if, for any two distinct triples $\langle X_1, Y_1, Z_1 \rangle \in M'$ and $\langle X_2, Y_2, Z_2 \rangle \in M'$, we have $X_1 \neq X_2$, $Y_1 \neq Y_2$, and $Z_1 \neq Z_2$.

The decision problem is, then, stated as follows: given a set M and an integer k, decide whether there exists a 3DM $M' \subseteq M$ such that $|M'| \geq k$.

Let us, now, turn to the reduction. Let M_i indicate the i-th triple in M and let $q = |M|$. Let $\Pi_1 = X \cup Y \cup Z \cup \Sigma$ and $\Pi_2 = \{M_i\} \cup \Sigma$, where the pairs (X_i, M_j) (resp., (Y_i, M_j), (Z_i, M_j)) are not constrained, i.e., any of them can possibly match.

For each triple M_i, Σ contains the set of symbols $\{c_{i_1} \dots c_{i_q}, t_{i_1} \dots t_{i_7}, m_{i_1} \dots m_{i_7}\}$. The set of constraints χ is constructed in such a way as to allow symbol identity only among the symbols in $\{c_{i_1} \dots c_{i_q}, t_{i_1} \dots t_{i_7}, m_{i_1} \dots m_{i_7}\}$.

Then, starting from M, we build the following strings:

$$
s_1 = \Phi_1 \ \ c_{1_1} \ c_{1_2} \ \cdots \ c_{1_q} \ \ \Phi_2 \ \ c_{2_1} \ c_{2_2} \ \cdots \ c_{2_q} \ \cdots \ \Phi_q \ \ c_{q_1} \ c_{q_2} \ \cdots \ c_{q_q}
$$

$$
s_2 = \Psi_1 \ \ c_{1_1} \ c_{1_2} \ \cdots \ c_{1_q} \ \ \Psi_2 \ \ c_{2_1} \ c_{2_2} \ \cdots \ c_{2_q} \ \cdots \ \Psi_q \ \ c_{q_1} \ c_{q_2} \ \cdots \ c_{q_q}
$$

where, for each triple M_i, the "blocks" Φ_i and Ψ_i have the form:

$$t_{i_1} \;\; X_i \;\; t_{i_2} \;\; t_{i_3} \;\; t_{i_4} \;\; Y_i \;\; t_{i_5} \;\; t_{i_6} \;\; t_{i_7} \;\; Z_i$$

$$M_i \;\; m_{i_1} \;\; m_{i_2} \;\; M_i \;\; m_{i_3} \;\; m_{i_4} \;\; m_{i_5} \;\; M_i \;\; m_{i_6} \;\; m_{i_7}$$

To clarify this construction, consider the set $M = \{\langle X_1, Y_2, Z_2\rangle, \langle X_2, Y_1, Z_1\rangle, \langle X_3, Y_2, Z_3\rangle, \langle X_3, Y_3, Z_4\rangle\}$. The first triple generates the following portions of strings:

$$s_1 = t_{1_1} \;\; X_1 \;\; t_{1_2} \;\; t_{1_3} \;\; t_{1_4} \;\; Y_2 \;\; t_{1_5} \;\; t_{1_6} \;\; t_{1_7} \;\; Z_2 \;\; c_{1_1} \;\; c_{1_2} \;\; c_{1_3} \;\; c_{1_4} \;\; \cdots$$

$$s_2 = M_1 \;\; m_{1_1} \;\; m_{1_2} \;\; M_1 \;\; m_{1_3} \;\; m_{1_4} \;\; m_{1_5} \;\; M_1 \;\; m_{1_6} \;\; m_{1_7} \;\; c_{1_1} \;\; c_{1_2} \;\; c_{1_3} \;\; c_{1_4} \;\; \cdots$$

Observe that, without any edit operation on s_1 and s_2, the $c_{i_1} \cdots c_{i_q}$ blocks match, whereas all the other ones (each consisting of 10 symbols) do not match. As a consequence, without any edit operation, the distance between the two strings is $d = 10 \cdot q$. For instance, in the previous example, the initial distance is $d = 40$.

Assume, now, that we are interested in computing $\mathcal{L}_{\langle \pi_1, \pi_2, \chi\rangle}(s_1, s_2)$, when $\pi_1 = 1$ and $\pi_2 = 3$. In other words, assume that each parameter in s_1 can match with at most one parameter in s_2 (this corresponds to say that each X_i (resp., Y_i, Z_i) can be selected in at most one triple), whereas each parameter in s_2 can match with up to 3 parameters in s_1 (this is needed to accommodate the triple $\langle X_i, Y_i, Z_i\rangle$ in M_i).

First we focus on the alignment of the blocks Φ_i and Ψ_i. It is easy to see that, if these blocks are considered in isolation, the only way to reduce their edit distance is to assign both X_i, Y_i, and Z_i to M_i in the matching schema and align them as follows:

$$t_{i_1} \;\; X_i \;\; t_{i_2} \;\; t_{i_3} \;\; t_{i_4} \;\; Y_i \;\; t_{i_5} \;\; t_{i_6} \;\; t_{i_7} \;\; Z_i \;\; - \;\; -$$

$$- \;\; M_i \;\; m_{i_1} \;\; m_{i_2} \;\; - \;\; M_i \;\; m_{i_3} \;\; m_{i_4} \;\; m_{i_5} \;\; M_i \;\; m_{i_6} \;\; m_{i_7}$$

In this case, the distance between these two blocks becomes 9. It is easy to check that, if at least one among X_i, Y_i, and Z_i is not associated with M_i in the matching schema, or a different alignment among blocks is carried out, the obtained distance is greater than or equal to 10. As a consequence, the corresponding choice is not convenient on the global perspective of finding the minimum edit distance.

Now, observe that the role of the blocks $c_{i_1} \cdots c_{i_q}$ is exactly to isolate the Φ_i-Ψ_i blocks; in fact, it is easy to verify that it will be never convenient to perform insertions or deletions within these blocks, because these operations would increase the overall distance.

In conclusion, $\mathcal{L}_{\langle \pi_1, \pi_2, \chi\rangle}(s_1, s_2)$ is obtained in correspondence of the matching schema that selects the maximum number of disjoint triples. The number of these triples can be determined by the decrease of the initial distance; indeed, each selected triple implies that distance decreases by 1. As a consequence, given a set M such that $|M| = q$, and an integer k, deciding whether there exists a 3DM $M' \subseteq M$ such that $|M'| \geq k$ corresponds to checking whether $\mathcal{L}_{\langle \pi_1, \pi_2, \chi\rangle}(s_1, s_2) \leq 10 \cdot q - k$, with $\pi_1 = 1$ and $\pi_2 = 3$. This implies that 3DM can be reduced to the computation of $\mathcal{L}_{\langle \pi_1, \pi_2, \chi\rangle}(s_1, s_2)$ over all valid matching schemas and, therefore, that this last task is NP-Hard. □

4.2 A lower bound L for $\mathcal{L}_{\langle \pi_1,\pi_2,\chi\rangle}(s_1, s_2)$

The intractability of the problem of computing $\mathcal{L}_{\langle \pi_1,\pi_2,\chi\rangle}(s_1, s_2)$ points out the need of heuristic approaches for its solution. In this context, it is highly useful to establish a lower bound L for $d^* = \mathcal{L}_{\langle \pi_1,\pi_2,\chi\rangle}(s_1, s_2)$ that could be computed in polynomial time; indeed, L could be used to evaluate the quality of the results returned by heuristic approaches.

First, we start with the definition of a lower bound L for d^* when $\pi_1 = 1$ and $\pi_2 = 1$; then, we generalize it to any value of π_1 and π_2. We point out that the computation of L is not a trivial task even for the case when $\pi_1 = 1$ and $\pi_2 = 1$; in fact, a straightforward lower bound could be easily set to 0, but a useful value for L should approximate d^* as much as possible.

Given two strings s_1 and s_2 over Π_1 and Π_2, respectively, our goal is to characterize d^* as precisely as possible by estimating the role, in the computation of d^*, of each pair of symbols (a, b) such that $a \in \Pi_1$ and $b \in \Pi_2$, without the need of computing the optimal matching schema preliminarily. Unfortunately, the role played by each pair (a, b) is not independent of the matchings of the other pairs of symbols included in the optimal matching schema. In particular, it is not possible to compute the exact role played by (a, b) without preliminarily fixing the whole matching schema.

As a consequence, our aim is to estimate d^* based on an estimation of the maximum number of potential matchings between pairs of symbols, given a maximum number of insertions and deletions (*indels*, in the following) performed on s_1 or s_2.

To simplify our presentation, in the following, we first assume that $|\Pi_1| = |\Pi_2| = |\Pi|$ and $len(s_1) = len(s_2) = l$; then, we extend the definition to the general case.

In order to describe our approach for the computation of L, we observe that d^* could be written as:

$$d^* = l + \delta^* - \mu^*$$

where δ^* is the number of indels present in the optimal alignment $\langle \bar{s}_1, \bar{s}_2\rangle$, and μ^* is the number of matches present in $\langle \bar{s}_1, \bar{s}_2\rangle$. In the following, we denote as $\langle \bar{s}_1, \bar{s}_2\rangle_\delta$ the optimal alignment which can be obtained by allowing at most δ indels.

Clearly, δ^* and μ^* are not known a priori. However, we can polynomially simulate the computation of d^* by estimating the number μ_δ of matches that can be obtained for increasing values δ of indels. The minimum value of the formula $l + \delta - \mu_\delta$ is, then, the best estimation for d^*. Formally:

$$L = \min_{\delta \in [0, l/2]} \{l + \delta - \mu_\delta\}$$

Now, given a maximum number δ of indels allowed in the alignment, we compute μ_δ by estimating the number of potential matches $\omega_\delta(a, b)$ for each pair of symbols (a, b) in the corresponding alignment $\langle \bar{s}_1, \bar{s}_2\rangle_\delta$ as follows:

$$\omega_\delta(a, b) = \min\left(\sum_{i=1}^{l} \kappa(a, b, s_1[i], s_2[i - \delta..i + \delta]), \sum_{i=1}^{l} \kappa(a, b, s_1[i - \delta..i + \delta], s_2[i])\right)$$

where $\kappa(a, b, substr_1, substr_2)$ returns 1 if a appears in the substring $substr_1$ and b appears in the substring $substr_2$; it returns 0 otherwise.

Observe that $\omega_\delta(a,b)$ is actually an overestimation of the true number of matches between a and b in $\langle \bar{s}_1, \bar{s}_2 \rangle_\delta$ because it is not guaranteed that a and b are aligned in $\langle \bar{s}_1, \bar{s}_2 \rangle_\delta$ every time they have been counted as a match.

Based on the values of ω_δ, we can compute μ_δ as follows. First, we construct a (complete) bipartite weighted graph $G_\delta = (V, U, E, \omega_\delta)$, where each vertex in V (resp. U) is a symbol of Π_1 (resp. Π_2). Then, we compute a *maximum weighted matching* M_δ on G_δ as follows:

$$\mu_\delta = \sum \omega_\delta(a,b) \ s.t. \ (a,b) \in M_\delta$$

Observe that the computation of the maximum weighted matching simulates the construction of the valid matching schema that allows the highest number of matches.

Once these two tasks have been performed, it is possible to state the following proposition:

Proposition 4.1 $L = \min_{\delta \in [0, l/2]} \{l + \delta - \mu_\delta\}$ is a lower bound for $d^* = \mathcal{L}_{\langle \pi_1, \pi_2, \chi \rangle}(s_1, s_2) = l + \delta^* - \mu^*$, that is $L \leq d^*$

Proof

To show that $L \leq d^*$, we must quantify the difference between the first and the second term of the inequality.

We start by characterizing μ_δ and we show that it is monotonically increasing. In fact, μ_δ is obtained by counting the co-occurrences of each pair of symbols that appear in a window of size $2 \cdot \delta + 1$, because the substrings considered by κ are $s_2[i - \delta..i + \delta]$ and $s_1[i - \delta..i + \delta]$, respectively. Clearly, enlarging the window does not produce a decrease of the co-occurrences.

Given the value δ^* corresponding to the optimal solution of δ, $\mu_{\delta^*} \geq \mu^*$. In fact, we have shown that the estimation of the matchings computed by μ_δ is, actually, an overestimation of the real matchings that can be obtained. Unfortunately, the real value of δ^* cannot be determined without deriving the exact solution.

Now, since $l + \delta^* - \mu^*$ is the minimum value that can be obtained from the exact solution, it follows that $\min_{\delta \in [0, l/2]} \{l + \delta - \mu_\delta\} \leq l + \delta^* - \mu^*$, since $\delta^* \in [0, l/2]$. This proves the proposition. □

Complexity analysis. The computation of L requires the computation of the minimum of $l + \delta - \mu_\delta$ for values of $\delta \in [0, l/2]$. As a consequence, the formula $l + \delta - \mu_\delta$ must be checked $O(l)$ times. Each check involves: *(i)* a maximum weighted matching computation over a bipartite graph composed of $2|\Pi|$ nodes; *(ii)* for each pair of symbols (a,b) in Π_1 and Π_2 (and, hence, for $O(|\Pi|^2)$ pairs), the computation of the weight $\omega_\delta(a,b)$. It is well known that the maximum weighted matching can be computed in $O(|\Pi|^3)$; the computation of $\omega_\delta(a,b)$ requires to span, for each of the l symbols in s_1 (resp., s_2) a portion of s_2 (resp., s_1) that can be as large as $O(l)$.

Summarizing, the worst case complexity of the lower bound computation is:

$$O(l \times (|\Pi|^3 + |\Pi|^2 \times l^2))$$

Generalization to the case of unequal alphabet sizes and string lengths. In this paragraph we analyze the generalization of the lower bound definition for strings with unequal alphabet sizes and string lengths.

First of all, the case in which alphabet sizes are not equal does not imply changes in the computation of the lower bound because, for each pair of symbols (a, b), the same computations for $\omega_\delta(a, b)$ must be carried out. Even the computation of the maximum weighted matching does not change: the graph would consist of a set of $|\Pi_1|$ nodes and a set of $|\Pi_2|$ nodes.

When the lengths of the strings are different, some more refined considerations on the possible number of indels in the optimal alignment, and on the length of the intervals to be checked in $\omega_\delta(a, b)$, must be carried out. In order to draw the discussion in an easier way, and without a loss of generality, we can assume that s_1 is always the shortest string, with $len(s_1) = l_m$ and $len(s_2) = l_M$.

First of all, consider that, given the different string lengths, and given the need of producing two equal length alignments $\bar{s}_1$ and $\bar{s}_2$, a number $\bar{\delta} = l_M - l_m$ of "gap filling" insertions on l_m will be always needed. As a consequence, the definition of d^* for the general case is:

$$d^* = l_m + \bar{\delta} + \delta^* - \mu^*$$

Observe that this is equivalent to write $d^* = l_M + \delta^* - \mu^*$. However, we will need to explicitly refer to $\bar{\delta}$ later on, defined as the additional indels to the "standard" ones. As a consequence, we prefer this more indel-oriented expression. In an analogous way, the definition of L is generalized to:

$$L = \min_{\delta \in [0, l_m/2]} \{l_m + \bar{\delta} + \delta - \mu_\delta\}$$

where,

$$\omega_\delta(a, b) = \min \left(\sum_{i=1}^{l_m} \kappa(a, b, s_1[i], s_2[i - \delta..i + \bar{\delta} + \delta]), \sum_{i=1}^{l_M} \kappa(a, b, s_1[i - \bar{\delta} - \delta..i + \delta], s_2[i]) \right)$$

Observe that, in the first term (that refers to the shorter string s_1), $\bar{\delta}$ is involved just in the upper bound for determining the substring of s_2, because it would be meaningless to delete more than $l_m/2$ symbols from s_1 for an optimal alignment. Analogously, in the second term (that refers to s_2), $\bar{\delta}$ is involved just in the lower bound for determining the subset of s_1, because it would be meaningless to insert more than $l_m/2$ symbols in s_2 for an optimal alignment.

It is clear enough that if $l_m = l_M$ then $\bar{\delta} = 0$ and all the formulas get back to the previous ones.

Extension to generic values of π_1 and π_2. The philosophy underlying the computation of L for generic values of π_1 and π_2 does not change with respect to the case when $\pi_1 = 1$ and $\pi_2 = 1$, examined in detail above. The only difference is that the computation of the maximum weight matching must be substituted by the computation of the proper subsets of Π_1 and Π_2 on the bipartite graph such that: *(i)* all the arcs within the subsets are considered in the computation of μ_δ, and *(ii)* μ_δ is maximum.

4.3 Heuristics for the computation of $\mathcal{L}_{\langle\pi_1,\pi_2,\chi\rangle}(s_1, s_2)$

After having proved that the problem of computing $\mathcal{L}_{\langle\pi_1,\pi_2,\chi\rangle}(s_1, s_2)$ is intractable, it appears reasonable to proceed with the definition of heuristics for solving it. This section aims at providing a contribution in this setting.

The basic idea underlying the proposed heuristics is that, if we were armed with the best matching schema, then the computation of the generalized edit distance between s_1 and s_2 could be done easily by means of dynamic programming. Thus, we may resort to heuristic methods in order to iteratively refine a starting matching schema while searching for the optimal one. Several heuristics could be adopted for this purpose, ranging from the ones based on local search, such as the *random-restart steepest ascent hill climbing algorithm* (hereafter, `HC`) or the *simulated annealing* (hereafter, `SA`), to those exploiting genetic algorithms based on *evolution strategy* (hereafter `ES`). In this section, we focus on the definition of these heuristics, whereas in Section 5 we report the results of several experiments evaluating them.

4.3.1 Random-restart steepest ascent hill climbing

Intuitively, at step 0, a starting matching schema M^0 is chosen, and $\mathcal{L}^{M^0}(s_1, s_2)$ is computed. At the generic iteration i, the neighbors $M^i_{\nu_j}$ of the current matching schema M^i are considered, and the distances $\mathcal{L}^{M^i_{\nu_j}}(s_1, s_2)$ are computed. A neighbor $M^i_{\nu_j}$ of a matching schema M^i is a perturbation of M^i that exchanges only one pair of symbols between two partitions of the same alphabet. The matching schema that guarantees the lowest distance is, then, chosen and set as the starting matching schema M^{i+1} for the next step. This activity stops when the edit distance cannot be further improved. When this happens, the current edit distance is returned as the result. To avoid repeated computations, a hash map registering the already verified matching schemas is exploited.

Finally, in order to increase the chances of finding the optimal alignment, a certain number of random restarts, each characterized by a new randomly selected matching schema, are carried out.

Since, at each step, we are interested in finding a matching schema that returns a distance $\mathcal{L}^{M^i_{\nu_j}}(s_1, s_2)$ lower than the current minimum d_{min}, in order to compute $\mathcal{L}^{M^i_{\nu_j}}(s_1, s_2)$ we resort to the approach presented by Landau and Vishkin in [33], which is able to compute the edit distance between s_1 and s_2 in $O(d_{min} \cdot max\{len(s_1), len(s_2)\})$, if this distance is less than d_{min}. The ideas illustrated above are formalized in the Algorithm `COMPUTE-F`, reported in Algorithm 1.

We observe that explicitly storing, comparing, and ordering matching schemas can be computationally heavy, from both the execution time and the space occupancy perspectives. In our solution, we take advantage of some peculiarities of matching schemas, which allow us to resort to a *virtual and compact* representation of them. This allows significant reductions of both the time and the space required to handle the manipulation of matching schemas.

In particular, given the pair of parameters π_1 and π_2, a valid configuration of a matching schema has always the form depicted in Figure 2(a), where a matching schema for $\pi_1 = 3$ and $\pi_2 = 2$ is used as an example. Observe that, if we change the order of the symbols in Π_1 and Π_2, instead of the content of the matrix, as represented in Figure 2(b), the representation of the matching schema still holds. As a consequence, a cheaper representation of matching schemas may simply resort to

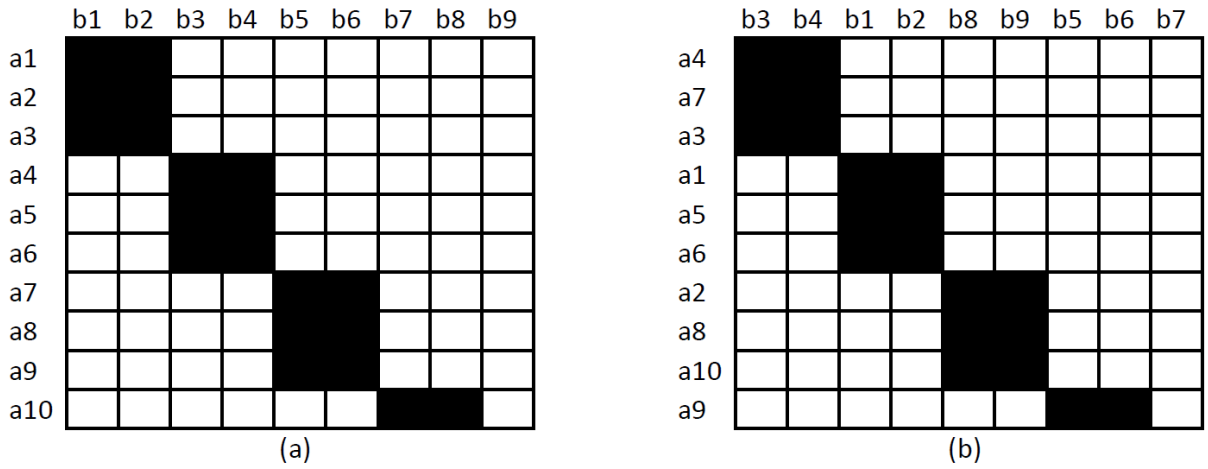

Figure 2: Examples of matching schemas for $\pi_1 = 3$ and $\pi_2 = 2$

the order of the symbols in Π_1 and Π_2, as they imply symbol subsets, matches, and mismatches. For instance, the matching schema of Figure 2(b) can be simply represented by the pair of ordered symbols $\Pi_1 = \{a4, a7, a3, a1, a5, a6, a2, a8, a10, a9\}$ and $\Pi_2 = \{b3, b4, b1, b2, b8, b9, b5, b6, b7\}$; in this case, the subsets of symbols $\{a4, a7, a3\}$ of Π_1 and $\{b3, b4\}$ of Π_2 match. With this representation at hand, changing the matchings of a symbol a_i in Π_1 with symbols in Π_2 simply translates with a change in the position of a_i in Π_1.

In our solution, we exploit this representation for initializing matching schemas, for computing neighbors and for storing checked matching schemas in the hash map. Since this representation is transparent to the actual usage of matching schemas, for the sake of presentation, we express operations on matching schemas as they were explicitly represented as matrices.

As for the used hash map, we rely on the *unordered_map* implementation offered by the *C++11 std library*. In particular, for each matching schema, the key for the hash map is obtained by the juxtaposition of the pair of ordered symbols of its representation, as described above. The selected implementation uses open hashing, and the elements in the hash map are organized into buckets, depending on their hash values.

One of the most interesting operations carried out in `COMPUTE-F` is the construction of the neighbors of a given matching schema M. This task is formalized in the function `Neighbors`, reported in Algorithm 2. In particular, given a starting matching schema M, its rows and columns are virtually swapped one by one in order to generate matching schemas that differ from M in only one symbol swap of subets.

4.3.2 Simulated Annealing

Simulated annealing is a very popular local search meta-heuristic adopted to address both discrete and continuous optimization problems [32, 23]. The main feature of `SA` is that of trying to escape local optima by moving towards worse solutions in the hope of reaching a global optimum. Generally, the aim of `SA` is to bring a system from an arbitrary initial state to one with the minimum possible energy.

```
Input  : two strings s_1 and s_2 over the alphabets Π_1 and Π_2, respectively;
         a set χ of constraints;
         three integers π_1, π_2, and T;
Output: L_⟨π_1,π_2,χ⟩(s_1, s_2);
Data   : two |Π_1| × |Π_2| matching schemas M and M';
         Checked: a hash map for tested matching schemas;
         improved: boolean;
         t, mindist, globaldist: integer
begin
    t = 0;
    initialize(M,π_1,π_2,χ);
    mindist = L^M(s_1, s_2);
    globaldist = mindist;
    improved = true;
    Checked = ∅;
    while improved do
        improved = false;
        N = Neighbors(M, χ);
        foreach M' in N and not in Checked do
            Checked=Checked ∪ M';
            if L^M'(s_1, s_2) < mindist then
                mindist = L^M'(s_1, s_2);
                improved = true;
                M = M';
            end
        end
        if not improved then
            if mindist < globaldist then
                globaldist = mindist;
                improved = true;
                t = 0;
            else if t < T then
                t = t + 1;
                improved = true;
                M = randomSelect(M,π_1,π_2,χ);
                mindist = L^M(s_1, s_2);
            end
        end
    end
    return globaldist;
end
```

Algorithm 1: The algorithm `COMPUTE-F` for the computation of $\mathcal{L}_{\langle\pi_1,\pi_2,\chi\rangle}(s_1, s_2)$ by means of `HC`

In particular, the algorithm behaves as follows. At each iteration, a new state is selected, its energy is computed and compared to the current one. If the computed energy is better, the system moves in the new state; otherwise, it decides to move in the new state anyway with a certain probability. The number of iterations depends on an initial parameter of `SA`, called *temperature*, which decreases with each iteration.

```
Function Neighbors (M, χ)
    Input  : a matching schema M over the alphabets Π1 and Π2, respectively;
             a set of constraints χ;
    Output: a set of matching schemas MS;
    Data   : n, m, r1, r2, c1, c2: integer;
    begin
        MS = ∅;
        for r1 = 0 to |Π1| do
            for r2 = r1 to |Π1| do
                swaprows(M, r1, r2);
                for c1 = 0 to |Π2| do
                    for c2 = c1 to |Π2| do
                        swapcolumns(M, c1, c2);
                        if isvalid(M,χ) then
                            MS = MS ∪ M;
                    end
                end
            end
        end
        return MS;
    end
```

Algorithm 2: The function `Neighbors` for the construction of the neighbors of a given matching schema M

In our context, a state is simply represented by a matching schema M and its energy is $\mathcal{L}^M(s_1, s_2)$. At each iteration, a neighbor M_r of the current matching schema M is selected and its energy $\mathcal{L}^{M_r}(s_1, s_2)$ is computed. In this case, a neighbor M_r of a matching schema M is a random perturbation of M.

The formalization of the solution with `SA` is detailed in Algorithm 3. In particular, it starts by setting the *temperature* parameter as $[(len(s_1) + len(s_2) \cdot (|\Pi_1| + |\Pi_2|)]$; this is the value also used in our experiments. Then, a first initial state (i.e., a matching schema M_{curr}) is determined according to π_1, π_2 and χ, and its energy (i.e., $\mathcal{L}^{\mathcal{M}_{\text{curr}}}(s_1, s_2)$) is computed. Until the temperature decreases, the current matching schema is randomly perturbed to generate a new matching schema. If this returns a better solution w.r.t. the current one, we move to the new state of the system; otherwise, the probability function `move`() is used to decide whether to move to the new state or not.

If we moved to the new state, the obtained distance is updated accordingly. Eventually, the temperature decreases to zero and the best distance, according to `SA`, is returned.

4.3.3 Evolution Strategy

Evolution Strategies [5, 16, 45] belong to the category of Genetic Algorithms and are inspired by the theory of evolution through natural selection. In particular, this heuristic is based on species-level processes of evolution, like hereditary and variation.

The aim of an Evolution Strategy (`ES`) algorithm is to improve the quality of a collection of candidates through the generation of descendants from parents by mutation operators, guided by an

```
Input  : two strings s_1 and s_2 over the alphabets Π_1 and Π_2, respectively;
         a set χ of constraints;
         two integers π_1, π_2;
Output: L_⟨π_1,π_2,χ⟩(s_1, s_2);
Data   : two |Π_1| × |Π_2| matching schemas M_curr and M_rand;
         temp, d_curr, d_rand, d*: integers;
begin
    temp = (len(s_1) + len(s_2))(|Π_1| + |Π_2|);
    initialize(M_curr,π_1,π_2,χ);
    d* = ∞;
    d_curr = L^{M_curr}(s_1, s_2);
    while temp > 0 do
        M_rand = randomPerturbation(M_curr, π_1, π_2, χ);
        d_rand = L^{M_rand}(s_1, s_2);
        if d_rand < d_curr or move() then
            d_curr = d_rand;
            M_curr = M_rand;
            if d_curr < d* then
                d* = d_curr;
            end
        end
        temp = temp −1;
    end
    return d*;
end
```

Algorithm 3: The algorithm for the computation of $\mathcal{L}_{\langle\pi_1,\pi_2,\chi\rangle}(s_1, s_2)$ by means of Simulated Annealing (`SA`)

objective function. Several variants of `ES` exist. In this paper, we consider the $(\mu + \lambda)$-`ES`, where μ is the number of candidate solutions in the parent generation, and λ is the number of candidate solutions obtained from the parent generation. Then, at each generation, the best μ individuals are kept from the λ candidates and their parents. An important parameter characterizing the algorithm is the number of generations considered for the evolution of the group.

Our implementation of `ES` for $\mathcal{L}_{\langle\pi_1,\pi_2,\chi\rangle}(s_1, s_2)$ is shown in Algorithm 4. Here, each individual is a matching schema M; it is efficiently represented as described in Section 4.3.1. The objective function used to select the best individuals in a generation is just $\mathcal{L}^M(s_1, s_2)$. Finally, a variant of the `Swap2` mutator is used as mutation operator; it selects two random elements of the matching schema and swaps them. The algorithm stops when the given number of generations `ngen` is reached and returns the best individual.

5 Experiments

In this section, we evaluate the applicability of the proposed framework and the effectiveness of the proposed heuristics for computing $\mathcal{L}_{\langle\pi_1,\pi_2,\chi\rangle}(s_1, s_2)$. In particular, first we compare the heuristics presented in this paper, namely `HC`, `SA`, and `ES`, in such a way as to determine their pros and

```
Input  : two strings s_1 and s_2 over the alphabets Π_1 and Π_2, respectively;
         a set χ of constraints;
         five integers π_1, π_2, μ, λ and ngen;
Output: L_⟨π1,π2,χ⟩(s_1, s_2);
Data   : P, C: set of individuals;
         c, p, p*: individuals;
         cgen: integer;
begin
    for i = 0 to μ do
        p = randomIndividual(π1,π2,χ);
        addTo(p, P);
    end
    cgen = 0;
    while cgen < ngen do
        C = ∅;
        for j = 0 to λ do
            c = randomSelectionFrom(P);
            applyMutation(c);
            setValue(c, L^c(s_1, s_2));
            addTo(c, C);
        end
        P = getBestIndividuals(C, P);
        cgen = cgen + 1;
    end
    p* = getBestIndividual(P);
    return getValue(p*);
end
```

Algorithm 4: The algorithm for the computation of $\mathcal{L}_{\langle \pi_1,\pi_2,\chi\rangle}(s_1, s_2)$ by means of Evolution Strategy (`ES`)

cons. Then, we provide a wide comparison between our approach and other kinds of heuristics. Due to space constraints, we present only a relevant subset of the obtained results. The interested reader can find the complete data sets, the executable codes and the obtained results at the address `https://www.mat.unical.it/terracina/kbs-mped/`.

5.1 Comparison of `HC`, `SA`, and `ES`

In our campaign, we considered two main aspects, namely: *(i)* the reliability and *(ii)* the execution time of the heuristics of interest. To precisely evaluate the reliability, we compared the results returned by the involved heuristics on a set of test data, pre-labeled with the exact value of $\mathcal{L}_{\langle \pi_1,\pi_2,\chi\rangle}(s_1, s_2)$ for them as computed by applying the approach `EX`, which, as described in Section 4, returns the exact solution. As we pointed out in that section, `EX` is unfeasible for large values of $|\Pi_i|$ and π_i, and, consequently, we performed this test only for small data sets. Then, in order to test the reliability of our approach on larger data sets, we carried out two further evaluations. The former considers, as the reference value for $\mathcal{L}_{\langle \pi_1,\pi_2,\chi\rangle}(s_1, s_2)$, the estimation provided by the lower bound; the latter assumes, as the reference value for $\mathcal{L}_{\langle \pi_1,\pi_2,\chi\rangle}(s_1, s_2)$, the lowest one returned by the three heuristics in each

test. All details are provided below. The analysis of the execution times has the twofold purpose of measuring the gain of the heuristics with respect to the exact solution and of verifying their actual applicability to real cases.

In the following, first we describe the exploited dataset and we discuss the values of the heuristics parameters; then, we illustrate the results of our evaluations.

5.1.1 Dataset

Each element of the dataset is a pair of strings (s_1, s_2). Without loss of generality, and to simplify our presentation, for each pair (s_1, s_2) we made the following assumptions:

- $len(s_1) = len(s_2) = len(s)$;
- $|\Pi_1| = |\Pi_2| = |\Pi|$;
- $\pi_1 = \pi_2 = \pi$.

To process a large corpus of tests for the comparison with `EX`, we generated test instances for each combination of the following values of $len(s)$, $|\Pi|$, and π:

- $len(s) = \{50, 100, 200, 350, 500\}$;
- $|\Pi| = \{3..10\}$;
- $\pi = \{1..4\}$ and such that $\pi < \left\lceil \frac{|\Pi|}{2} \right\rceil$.

Furthermore, to test the reliability of our approach for larger instances, we generated further ones with the following properties:

- $len(s) = \{1000, 2000, 3500\}$;
- $|\Pi| = \{8, 10, 12, 14, 16\}$;

5.1.2 Parameters for the heuristics

We performed all our tests with specific values of the heuristic parameters. Whenever not differently specified, the following parameter values have been used:

- as for `HC`, the number of restarts, denoted by the input parameter T in Algorithm 1, was set to 3;
- as for `SA`, the initial value of the *temperature* was set to $[(len(s_1) + len(s_2)) \cdot (|\Pi_1| + |\Pi_2|)]$;
- as for `ES`, μ was set to 5, λ was set to 20 and the maximum number of generations was set to 20.

		$len(s)$														
π	$\lvert\Pi\rvert$	50			100			200			350			500		
		P_{HC}	P_{SA}	P_{ES}	P_{HC}	P_{SA}	P_{ES}	P_{HC}	P_{SA}	P_{ES}	P_{HC}	P_{SA}	P_{ES}	P_{HC}	P_{SA}	P_{ES}
1	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99	1.00	1.00	1.00	1.00
	5	0.98	0.98	0.99	0.96	0.96	0.97	0.99	0.99	0.98	0.99	0.99	1.00	0.99	0.99	1.00
	6	1.00	1.00	1.00	0.97	0.97	0.98	1.00	1.00	1.00	0.98	0.98	0.98	0.97	0.97	0.98
	7	1.00	1.00	1.00	0.97	0.97	0.99	0.96	0.96	0.98	1.00	1.00	1.00	0.97	0.97	0.99
	8	0.93	0.93	0.96	0.98	0.98	0.99	0.96	0.96	0.98	0.97	0.97	0.99	1.00	0.98	1.00
	9	1.00	0.93	0.99	1.00	1.00	1.00	1.00	0.98	1.00	0.98	0.98	1.00	0.99	0.99	1.00
	10	0.93	0.93	0.97	0.95	0.95	0.97	0.97	0.97	0.98	1.00	0.97	0.99	0.97	0.97	0.99
2	5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99	1.00
	6	1.00	1.00	1.00	1.00	1.00	1.00	0.96	0.96	0.98	0.99	0.99	0.99	0.97	0.97	0.99
	7	1.00	1.00	1.00	0.98	0.98	1.00	0.97	0.97	0.99	0.99	0.98	0.99	1.00	0.98	1.00
	8	1.00	1.00	1.00	0.99	0.98	0.99	0.98	0.98	0.99	1.00	0.98	1.00	0.99	0.98	1.00
	9	1.00	0.98	1.00	1.00	0.99	1.00	0.99	0.99	1.00	0.99	0.97	0.99	0.98	0.97	1.00
	10	0.97	0.96	0.98	0.99	0.98	1.00	0.98	0.97	0.99	1.00	0.96	1.00	0.99	0.96	1.00
3	7	0.93	0.93	0.95	0.94	0.94	0.96	1.00	1.00	1.00	0.99	0.97	0.99	0.99	0.98	1.00
	8	0.98	0.98	0.99	0.98	0.97	0.99	1.00	0.98	1.00	0.99	0.98	1.00	0.99	0.98	0.99
	9	1.00	0.96	1.00	0.99	0.95	1.00	0.99	0.96	0.99	1.00	0.97	1.00	1.00	0.96	1.00
	10	0.97	0.95	0.98	0.97	0.96	0.99	0.98	0.95	0.99	0.99	0.98	1.00	0.99	0.97	0.99

Table 1: Values of P_{HC}, P_{SA}, and P_{ES} for different values of π, $|\Pi|$ and $len(s)$

5.1.3 Reliability

As previously pointed out, to measure the reliability of `HC`, `SA`, and `ES`, we compared the results provided by them with those returned by `EX`. For each test, we carried out ten executions of the heuristics and averaged the obtained solutions.

Given a heuristic `H` (where `H` can be `HC`, `SA`, `ES`), we measured the Precision of the solutions provided by `H` as follows. Let d_{EX} be the exact solution for an instance of $\mathcal{L}_{\langle \pi_1, \pi_2, \chi \rangle}(s_1, s_2)$, and let d_H be the (average) solution returned by `H`. The Precision P_H is computed as:

$$P_H = 1 - \frac{d_H - d_{EX}}{d_{EX}}$$

Table 1 shows P_{HC}, P_{SA}, and P_{ES} for the configurations that we tested. Note that a Precision of 1.00 means that the corresponding heuristics provided exactly the value d_{EX}.

From the analysis of Table 1 we observe that, even with long sequences or large alphabets, P_{HC}, P_{SA}, and P_{ES} are always very high. This is encouraging and allows us to expect a very high reliability of the two heuristics in real application cases. Among these results, `ES` shows slightly better results overall, whereas `SA` shows the lowest Precision in some cases.

As previously pointed out, in a second group of experiments, we used the lower bound, computed as described in Section 4.2, as the reference value for $\mathcal{L}_{\langle \pi_1, \pi_2, \chi \rangle}(s_1, s_2)$. In particular, given the lower bound d_L computed for $\mathcal{L}_{\langle \pi_1, \pi_2, \chi \rangle}(s_1, s_2)$, and the (average) solution d_H returned by one of the

	$len(s)$								
$\|\Pi\|$	1000			2000			3500		
	P^l_{HC}	P^l_{SA}	P^l_{ES}	P^l_{HC}	P^l_{SA}	P^l_{ES}	P^l_{HC}	P^l_{SA}	P^l_{ES}
8	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84
10	0.82	0.81	0.82	0.80	0.78	0.81	0.82	0.80	0.81
12	0.82	0.80	0.83	0.82	0.81	0.82	0.81	0.79	0.82
14	0.80	0.79	0.81	0.81	0.80	0.81	0.81	0.80	0.82
16	0.80	0.79	0.80	0.82	0.81	0.83	0.82	0.79	0.83

Table 2: Values of P^l_{HC}, P^l_{SA}, and P^l_{ES} for different values of $|\Pi|$ and $len(s)$

	$len(s)$								
$\|\Pi\|$	1000			2000			3500		
	P^m_{HC}	P^m_{SA}	P^m_{ES}	P^m_{HC}	P^m_{SA}	P^m_{ES}	P^m_{HC}	P^m_{SA}	P^m_{ES}
8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
10	1.00	0.98	1.00	0.99	0.96	1.00	1.00	0.97	0.99
12	0.99	0.97	1.00	1.00	0.99	1.00	0.99	0.97	1.00
14	0.99	0.98	1.00	1.00	0.98	1.00	0.99	0.98	1.00
16	1.00	0.97	1.00	0.99	0.98	1.00	0.99	0.97	1.00

Table 3: Values of P^m_{HC}, P^m_{SA}, and P^m_{ES} for different values of $|\Pi|$ and $len(s)$

considered heuristics `H`, the Precision P^l_H for this case is defined as:

$$P^l_H = 1 - \frac{d_H - d_L}{d_L}$$

Table 2 shows P^l_{HC}, P^l_{SA}, and P^l_{ES} for these tests. The corresponding results can be read from two perspectives. First, even for bigger data sets, Precision looks satisfactory, provided that it is almost always greater than 80%. However, if we read these values together with the ones shown in Table 1, we may interpret them on the side of the tightness of the lower bound. In particular, we may state that the lower bound is generally not lower than 20% of the exact value, then showing a good approximation.

Finally, to better test the differences between the heuristics into consideration, we also carried out a set of experiments where we used, as reference value for $\mathcal{L}_{\langle \pi_1, \pi_2, \chi \rangle}(s_1, s_2)$, the lowest one computed by the three heuristics. This can also be considered as an upper bound for the exact solution. In particular, given the lowest value d_{min} computed by `HC`, `SA`, and `ES` for a given test, and given the (average) solution d_H returned by one of the considered heuristics `H`, we computed the Precision P^m_H as:

$$P^m_H = 1 - \frac{d_H - d_{min}}{d_{min}}$$

Table 3 shows P^m_{HC}, P^m_{SA}, and P^m_{ES} for this case. In this table, a Precision equal to 1 for a certain heuristic `H` indicates that it always returned the lowest value for $\mathcal{L}_{\langle \pi_1, \pi_2, \chi \rangle}(s_1, s_2)$ in all the runs for that configuration.

From the analysis of this table, it is possible to observe that there are only small differences among

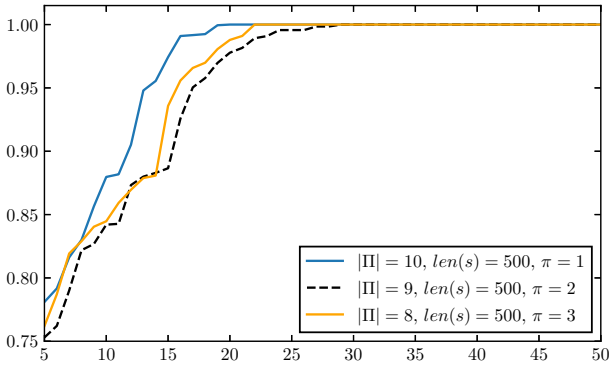

Figure 3: Precision P_{ES} against the number of generations for different combinations of $|\Pi|, len(s)$ and π

the three heuristics, with `SA` showing slightly worst results overall and `ES` showing almost always the best results.

5.1.4 A deeper analysis on the reliability of `ES`

In the previous section, we have shown that, overall, `ES` presents better performances than `HC` and `SA` in terms of Precision. Here, we better analyze this heuristic, based on one of the most relevant parameters for it, namely the number of generations.

As a matter of fact, a high number of generations corresponds to a wide search space; this implies a high probability of obtaining the best solution. However, a high number of generations implies a high execution time required to explore the search space. As a consequence, a proper tradeoff between Precision values and running times must be found.

To carry out this analysis we considered three of the most complex configurations for which the exact solution was known. Then, for each of them, we ran `ES` with increasing numbers of generations, from 5 to 50. For each test we carried out several runs and averaged the obtained results. Then, we computed Precision with respect to the best exact solution.

The obtained results are shown in Figure 3. From the analysis of this figure, it is straightforward to observe that a low number of generations (around 5) cannot guarantee a good level of Precision. However, this parameter quickly increases when the number of generations increases and asymptotically tends to 1. From 20 generations on, the improvement of Precision against the increase of the number of generations is marginal. As a consequence, from this analysis, it is possible to conclude that 20 represents the minimum number of generations necessary to obtain satisfactory results in terms of Precision. In the next section, we show that this value is also satisfactory in terms of execution time.

5.1.5 Execution time

In order to evaluate the temporal behavior of the implemented heuristics we designed two main experiments focusing on the variations of $|\Pi|$ and $len(s)$, respectively.

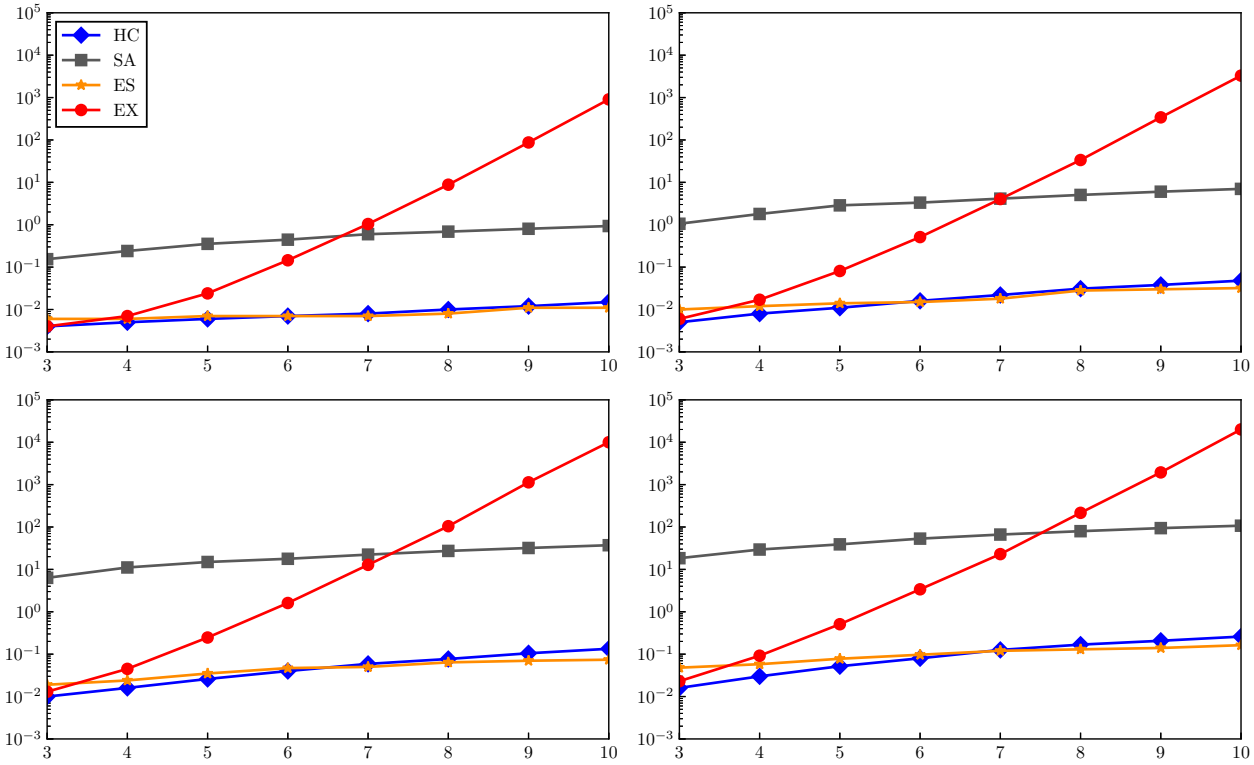

Figure 4: Execution time (in seconds) of `EX`, `HC`, `SA`, and `ES` against $|\Pi|$. The four graphs, from top to bottom and from left to right, indicate the results for $len(s) = 100$, 200, 350 and 500, respectively

We start by showing the results of the experiment centered on the variation of $|\Pi|$. For each value of this parameter belonging to the integer interval $[3, 10]$, we computed the running time necessary to obtain $\mathcal{L}_{\langle 1,1,\emptyset\rangle}(s_1, s_2)$ by applying `EX`, on the one hand, and `HC`, `SA`, and `ES`, on the other hand. Execution times were measured in seconds. We carried out the same computation for each value of $len(s)$ belonging to the set $\{100, 200, 350, 500\}$. Figure 4 shows the obtained results. Here, for each value of $len(s)$, we show the increase of the execution time against $|\Pi|$.

From the analysis of this figure, it is easy to see that, as expected, `EX` shows an exponential increase of its running time; therefore, its application becomes soon unrealistic. `SA` is always slower than `HC` and `ES` and, in some cases, even slower than `EX`. This is due to the intrinsic nature of this heuristic, which may require a certain number of unnecessary iterations. `HC` is always faster than `EX` and always below 1 second. `ES` can be slightly slower than `EX` and `HC` for low values of $|\Pi|$; this is mainly due to the number of generations that must be checked anyway. However it scales better than `HC`, and better than all the other heuristics, showing the fastest execution times for high values of $|\Pi|$.

The second experiment focuses on the variation of the execution time of `EX`, `HC`, `SA`, and `ES` against $len(s)$. In particular, for each value of $len(s)$ belonging to the set $\{50, 100, 200, 350, 500\}$, we computed the running time necessary to obtain $\mathcal{L}_{\langle 1,1,\emptyset\rangle}(s_1, s_2)$ for `EX`, `HC`, and `SA`. In this case, we considered six different values of $|\Pi|$, i.e., $|\Pi| \in \{5..10\}$. Therefore, we obtained six different graphs. The corresponding results are shown in Figure 5.

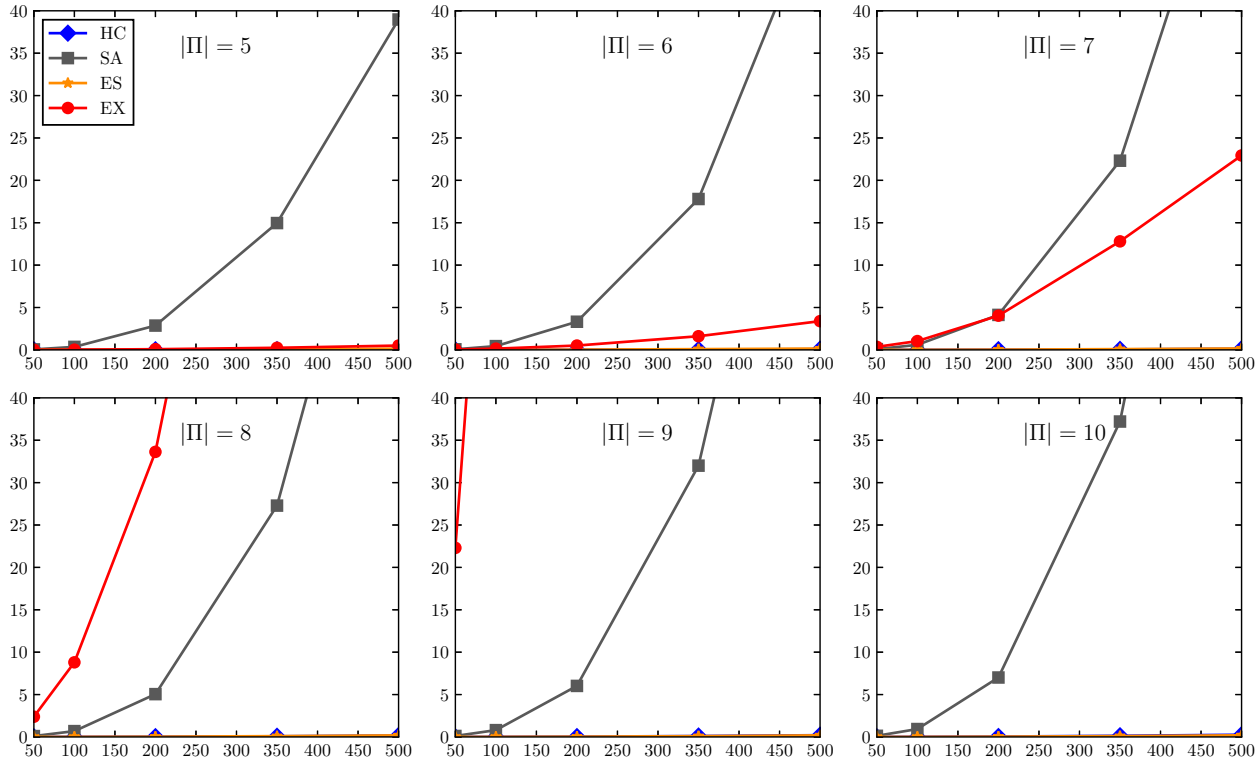

Figure 5: Execution time (in seconds) of `EX`, `HC`, `SA` and `ES` against $len(s)$. The six graphs, from top to bottom and from left to right, indicate the result for $|\Pi| \in \{5..10\}$

From the analysis of this figure, it is easy to see that an increase of l and an increase of $|\Pi|$ lead to a steep increase of the execution time of `EX`, which becomes out of scale already for $len(s) = 50$ with $|\Pi| = 10$. The execution time of `HC` and `ES` are basically flat and always below 1 second for each configuration; this make `HC` and `ES` far more perfoming than `SA`.

As a closing remark, considering the results of both reliability and execution time, we can conclude that `HC` and `ES` perform similarly and both better than `SA`, with a slight preference for `ES` in terms of both Precision and execution time.

Interestingly, as it was shown in [11], `ES` can be easily parallelized, with the possibility of obtaining significant improvements, in terms of performances, on parallel architectures.

5.2 Analysis of different heuristics

In this section, we carry out a wider analysis, comparing `HC`, `SA`, and `ES` with several other kinds of heuristics. In order to easily test these last ones, we used a tool called HeuristicLab (HL for short). HL is an environment in which different already implemented heuristics can be used; it allows the end user to focus on problem design rather than on heuristics implementation [49].

In particular, we tested the following algorithms available in HL: *local search*, *simulated annealing*, *genetic algorithm*, *offspring selection genetic algorithm*, *genetic algorithm with an age-layered popula-*

tion structure, and *offspring selection evolution strategy*. The aim of these tests is to check whether there exists a category of heuristics more promising than `ES` and `HC`; we focused on the capability of reaching the lowest value of MPED in the lowest number of iterations.

HL allows the setting of a certain number of parameters for each heuristic. In particular:

- As for *local search*, we set the sample size to $|\Pi_1| + |\Pi_2|$.
- As for (the HL version of) *simulated annealing*, we set the number of inner iterations to $|\Pi_1|+|\Pi_2|$, the start temperature to 100, and we used the exponential discrete double-value modifier as annealing operator.
- As for *genetic algorithm*, we set the population size to $10(|\Pi_1|+|\Pi_2|)$, we selected {swap2, swap3} as mutators, we chose {partially matched, cyclic} as crossover operators and {proportional, with windowing} as selectors, then we set the mutation probability to 5% and the elites to 1.
- As for *offspring selection genetic algorithm*, parameters were the same as the genetic algorithm and we set the maximum selection pressure and the number of selected parents to 200, the comparison factor to 0, the lower bound comparison factor and the upper bound success ratio to 1, and the offspring selection before mutation to false.
- As for the *genetic algorithm with ALPS*, we set the number of layers to 10, the selector as generalized rank with pressure 4 and the plus selection to false; the other parameters were the same as the genetic algorithm.
- As for *offspring selection evolution strategy*, we set the maximum selection pressure to 200, the comparison factor to 0.5, the success ratio to 1, the elites to 1, the selected parents to 40 and the plus selection to true, with the values of population size and mutators identical to the corresponding ones of the genetic algorithm.

We considered pairs of randomly generated strings. The varied parameters are the *alphabet cardinalities* $|\Pi_1|$ and $|\Pi_2|$ and the string lengths $len(s_1)$ and $len(s_2)$. Furthermore, we added a correlation degree to each pair of strings. In particular, we considered the following parameter values:

- $len(s) = 256$;
- $|\Pi| = \{8, 12, 16\}$;
- $\pi = 1$;
- degree of added correlation: $\{0, 0.5, 1\}$.

For each combination of parameters, we generated one instance of the problem. For each of these instances, we applied the heuristics we wanted to investigate. To take the result stochasticity into account, we repeated the execution of each heuristic 10 times on the same problem instance.

Due to space constraints, we discuss here only the results obtained for $|\Pi| = 12$ and a degree of added correlation equal to 0.5; these are shown in Figure 6. In this figure, all the algorithm names

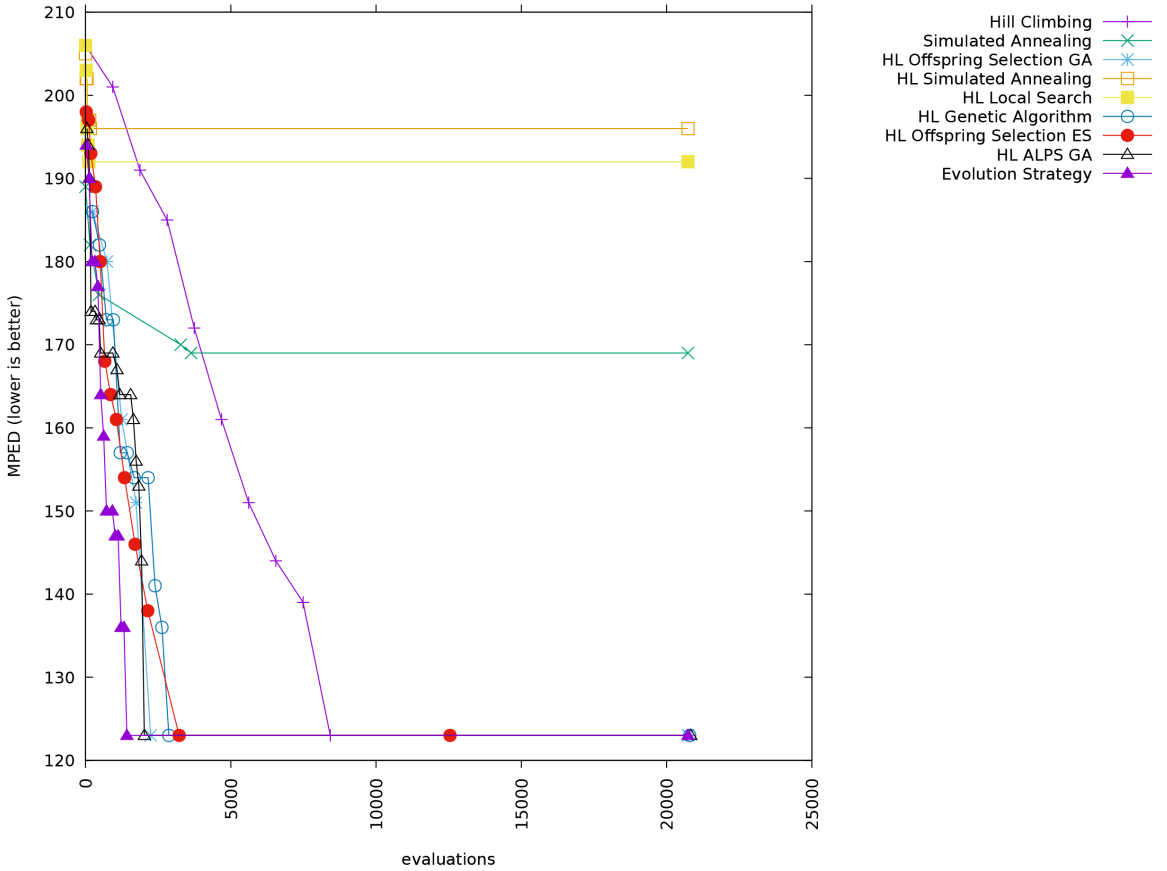

Figure 6: Results obtained by applying several heuristics when $|\Pi|$ is set to 12 and the degree of added correlation is set to 0.5

starting with HL refer to the implementations provided by Heuristic Lab. The other ones are the algorithms implemented in this paper. The interested reader can find all the other results at the address: `https://www.mat.unical.it/terracina/kbs-mped/`.

From the analysis of Figure 6, it is possible to observe that, even if `HC` performs well, its results are further improved again by `ES`, which also shows the overall best performance. The results of `ES` are generally better than the ones of all the HL heuristics.

6 Applications of $\mathfrak{F}_{\mathcal{L}}$

In this section, we present three cases where we applied $\mathfrak{F}_{\mathcal{L}}$. The purpose of this presentation is twofold. In fact, it proves that there are very different real cases that can highly benefit from the specific metric introduced in this paper. Furthermore, as an even more interesting fact, it gives an idea of the enormous potentialities of our framework, especially if we generalize all the possible string metrics mentioned in Section 3.1 in a way analogous to what we have done for the edit distance, which MPED derives from.

6.1 Application of MPED in Wireless Sensor Networks

A Wireless Sensor Network (hereafter, WSN) is a network of multi-functional and low-cost sensors, characterized by low energy consumption. Examples of sensors adopted in WSNs are those measuring temperature, humidity, light, noise, electric current, voltage and power.

WSNs are becoming increasingly pervasive thanks to the increasingly powerful technology, which they are based on [2]. However, owing to their enormous complexity and heterogeneity, new efficient techniques to manage them [18, 19] are necessary.

In this context, one of the most challenging issues to address regards the automatic detection of anomalies [9], i.e., the identification of items, events or observations giving rise to suspicions. Anomalies may rise for several reasons. Think, for instance, of devices that run out of power or that deviate from their expected behavior.

In past literature, a lot of efforts were made for anomaly detection in *homogeneous* WSNs [55, 1, 52, 38]. The proposed techniques are generally based on mathematical or statistical computations applied on data streams and tailored to the characteristics of sensed data. Instead, anomaly detection in *heterogeneous* sensor networks has received less attention. Furthermore, the approaches conceived for homogeneous WSNs do not show good performance when applied in this new, and very different, scenario.

Our framework, when specialized to MPED (i.e., $\mathfrak{F}_{\mathcal{L}}$), can play a relevant role in this application case. As a matter of fact, in [12], we proposed an MPED-based approach to anomaly detection in WSNs. This approach consists of two phases. The former aims at training the WSN into examination under "normal" operating conditions. The latter applies the knowledge thus gained to identify potential anomalies. Intuitively, one of the novelties of this approach in this scenario consists in the fact that the training phase does not compute expected *values* for the involved sensors, but expected *correlations* among them. In particular, during the first phase, for each sensor, and for each hour of the day, our approach identifies the most correlated sensor (called mate). This is used as the reference one during the second phase. In fact, whenever the correlation between a sensor and a mate significantly changes, a potential anomaly is registered.

$\mathfrak{F}_{\mathcal{L}}$ plays a relevant role in this approach because the sensors to examine could be heterogeneous and, in this case, the classical edit distance would fail. In [12], we showed that our approach is capable of precisely identifying anomalies spanning over long periods, which are difficult to be detected by other methods. Furthermore, we proved that our approach presents not only a good quality in measuring signal relatedness, but also a good robustness to unexpected variations of signals, as well as a good sensitivity to signal noises.

6.2 Application of MPED to extract and characterize White Matter fiber-bundles of brain

In the investigation of the brain, the capability of extracting and visualizing White Matter (hereafter, WM) fibers plays a key role. For instance, the knowledge of these fibers can provide an important contribution to understand and predict the effects of several neurodegenerative pathologies [51]. Tractography is currently considered the most accurate method to perform this task [39].

Particular sets of fibers (called fiber-bundles) denote different WM structures [10]. Their investigation is extremely important for neurologists. However, in order to carry out this task, it is necessary to isolate subsets of fibers belonging to a certain WM region. The isolation activity is often carried out manually by expert neuroanatomists, who define the suitable criteria allowing a region of interest to be delineated and specific fiber-bundles to be isolated [36]. Clearly, this way of proceeding is time expensive and the definition of automatic, or at least semi-automatic, approaches is in order.

As a matter of fact, in the past, several automatic approaches to isolate WM fiber-bundles have been proposed (see, for instance, [54, 21]). Most of them are based on clustering. In particular, the clustering task is guided by the fiber layout in the three-dimensional space. Actually, approaches operating in this way suffer from several limitations. For instance, some of them require a complex and time consuming supervision performed by experts, who carry out it through a "try-and-check" activity.

Our framework specialized to MPED (i.e., $\mathfrak{F}_{\mathcal{L}}$) can provide an important contribution to address this issue. Indeed, in [47, 13], we proposed a semi-automatic model-guided approach to allow the extraction of WM fiber-bundles from a set of tractography streamlines. Our approach is based on a fiber-bundle model constructed with the support of an expert and representing an approximate shape of the fiber-bundle to extract. It models a fiber as a sequence of m voxels in the three-dimensional space. A color can be associated with each voxel, specifying its spatial orientation. As a consequence, a fiber can be represented as a sequence of colors, each expressed in the RGB color space. This last can be discretized in such a way that a fiber can be represented as a string.

In this way, the WM fiber-bundle extraction problem reduces to a string similarity one. MPED can provide a great contribution to address this problem. Indeed, thanks to it, the extraction of WM fiber-bundles can be performed by computing (through MPED) the dissimilarity between the model fiber-bundles and the real fibers and, then, by selecting the real fiber-bundles having a dissimilarity degree w.r.t. the model fiber-bundle less than a given threshold.

Interestingly, in this case, data are not heterogeneous in their original format, but the adoption of a string-based representation of WM fibers allows a complex problem involving multi-dimensional data to be reduced to a simple comparison of heterogeneous strings. The heterogeneity of these last ones is due to the fact that they derive from several data flows. These are very different but, on the other side, their heterogeneity is extremely precious because it guarantees a multi-view and complete representation of the event to investigate (see [47, 13, 14] for all details).

Extensive tests carried out on this scenario proved the goodness of this idea. Furthermore, they showed that an approach based on $\mathfrak{F}_{\mathcal{L}}$ can outperform existing and more specific algorithms.

6.3 Application of MPED to time series forecasting

Time series analysis is an important research area that is receiving an increasing interest. It comprises methods for data analysis conceived to extract meaningful statistics or hidden properties of data. Among others, time series forecasting aims at predicting future values based on both previously observed values and a model. The potential applications of forecasting methods span several areas, from economic trends to weather forecast and the prediction of user behavior.

Numerous kinds of approach have been exploited for carrying out this task; most of them are

based on statistical analysis, mathematical models and, recently, neural networks and deep learning (see [42, 8, 35, 25, 53], just to cite a few).

In this context, the MPED's capability of identifying hidden correlations between heterogeneous data streams may provide an important contribution; in fact, it introduces a completely new way of addressing the problem at hand.

Specifically, it would be particularly interesting to verify whether the capability of identifying strongly related heterogeneous data streams, of aligning them, and of singling out the best matching schemas between them can provide useful information to predict, with some probability, future values of a stream based on previous observations. We next sketch this novel idea, which we intend to further develop and test in some future work.

First of all, observe that two sequences with a very low MPED can be considered strongly related, as there exists an alignment between them, characterized by very few indels, and where most of the symbols of the first sequence match with corresponding ones of the second sequence. Given a target sequence T, for which we want to predict future values, a first step of this new approach would consist in finding a *coupled* sequence C, which (possibly for no apparent reasons) is strongly related to T. In particular, we want to exploit observations on C to predict, with some probability, future values of T. Then, we can compute some statistics by observing the past values of C and T, and, specifically, by analyzing potential "cause-effect" properties between symbols appearing in C at the time instant t and symbols appearing in T in the interval $[t+1, t+\delta]$.

In our framework this can be obtained quite simply by:

- computing the best matching schema between symbols in C and T (with the limitations introduced in the next items);
- obtaining the best alignment between C and T but limiting indels only to insertions on C; this actually imposes to consider only shifts forward in time of symbols (events) of C to be matched with symbols (events) of T[3];
- limiting δ to a fixed interval of interest $[1, \delta_{max}]$.

The previous setting has several consequences. Indeed, if a strong correlation between C and T is detected, then almost every symbol in C at any time instant t finds a match in T within the following $1+\delta_{max}$ time instants. Consequently, given a symbol σ in C at the time instant t, we can state that most probably, within the next $1+\delta_{max}$ time instants, a symbol μ of T, matching σ according to the best matching schema, will be present in T. This probability can be easily computed by counting the number of times σ and μ are actually aligned in the optimal alignment, for every pair of σ and μ. Observe that our approach allows the definition of the values of π_1 and π_2 such that each symbol in C might match more symbols in T. In this case, we can provide multiple potential values of T that can be found in $[t+1, t+\delta_{max}]$, each associated with a given probability.

Once all this data is computed, given a new symbol (event) in C, our approach can provide a list of potential values that could appear in T within the next $1+\delta_{max}$ time instants, along with the

[3]Observe that this can be easily implemented by limiting the possible "moves" in the dynamic programming implementation of MPED.

corresponding probabilities. As a consequence, a form of probabilistic forecasting can be carried out on T.

Consider, again, the context of sensor analysis. Assume that we have determined, through the approach described above, that the data stream of a temperature sensor (the target T) is strongly related to the one of a light sensor (the coupled sensor C). Then, given a certain value of light at the time instant t, we might provide the probability of finding a certain value of temperature measured by T within the next $1 + \delta_{max}$ time instants.

Obviously, high values of δ_{max} may increase the probability of finding frequent co-occurring matches between symbols in C and T; however, a high δ_{max} can reduce the significance of the prediction. Conversely, a value of δ_{max} too low may significantly decrease the chance of finding a sequence strongly correlated to T. As a general rule, δ_{max} should be set to the maximum number of time instants that are considered a reasonable period of uncertainty for obtaining the expected values.

7 Conclusion

In this paper, we have presented a framework aiming at generalizing existing string metrics in such a way as to make them suitable for application scenarios where involved strings could be based on heterogeneous alphabets. Our framework consists of a matching schema, which formalizes matches between symbols, and a generalized metric function, which abstracts the computation of string metrics taking the predefined matching schema into account.

We have provided an overview of the application of the proposed framework for generalizing some of the most important string comparison metrics. Then, to give a precise idea of how this last activity can be carried out, we have discussed in detail the generalization of the edit distance obtained by using the proposed framework. Specifically, we have defined the Multi-Parameterized Edit Distance (MPED), we have investigated its computational properties, we have proposed some solution algorithms for it, and, finally, we have presented our experiments conceived to evaluate these algorithms.

As a final contribution, we have mentioned two applications in which the proposed generalized metric has already proved its potential and one application where it seems really promising.

This paper should not be considered as an ending point. Instead, it is the starting point of future research efforts in this field. First, the generalization of the other standard metrics mentioned in Section 3.1 can pave the way to a wide variety of new application scenarios. Second, it appears challenging to investigate the simultaneous comparison of multiple strings and the identification of the best matching schema among them. In this context, it is not straightforward to define the meaning of a globally optimal matching schema. Last, but not the least, an interesting research line consists in the application of our generalized framework to continuous data streams. In this case, a simple repeated computation of the metric at hand for each new incoming symbol would be computationally unfeasible. As a consequence, new forms of continuous computation of metrics are in order.

Acknowledgments

This work was partially supported by the Italian Ministry for Economic Development (MISE) under the project "Smarter Solutions in the Big Data World", funded within the call "HORIZON2020" PON I&C 2014-2020, and by the Department of Information Engineering at the Polytechnic University of Marche under the project "A network-based approach to uniformly extract knowledge and support decision making in heterogeneous application contexts" (RSAB 2018).

References

[1] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[3] A. Amir and I. Nor. Generalized function matching. *J. Discrete Algorithms*, 5(3):514–523, 2007.

[4] A. Apostolico, Péter L. Erdös, and M. Lewenstein. Parameterized matching with mismatches. *J. Discrete Algorithms*, 5(1):135–140, 2007.

[5] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996.

[6] B.S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *STOC*, pages 71–80. ACM, 1993.

[7] B.S. Baker. Parameterized diff. In *SODA*, pages 854–855. ACM/SIAM, 1999.

[8] J. Bi, H. Yuan, L. Zhang, and J. Zhang. Sgw-scn: An integrated machine learning approach for workload forecasting in geo-distributed cloud data centers. *Inf. Sci.*, 481:57–68, 2019.

[9] H.H.W.J. Bosman, A. Liotta, G. Iacca, and H.J. Wortche. Anomaly detection in sensor systems using lightweight machine learning. In *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*, pages 7–13. IEEE, 2013.

[10] M. Catani and M.T. de Schotten. A diffusion tensor imaging tractography atlas for virtual in vivo dissections. *Cortex*, 44(8):1105–1132, 2008.

[11] F. Cauteruccio, D. Consalvo, and G. Terracina. High performance computation for the multi-parameterized edit distance. In *26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 567–574, 2018.

[12] F. Cauteruccio, G. Fortino, A. Guerrieri, A. Liotta, D.C. Mocanu, C. Perra, G. Terracina, and M. Torres Vega. Short-long term anomaly detection in wireless sensor networks based on machine learning and multi-parameterized edit distance. *Information Fusion*, 52:13–30, 2019.

[13] F. Cauteruccio, C. Stamile, G. Terracina, D. Ursino, and D. Sappey-Marinier. An automated string-based approach to white matter fiber-bundles clustering. In *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*, pages 1–8. IEEE, 2015.

[14] F. Cauteruccio, C. Stamile, G. Terracina, D. Ursino, and D. Sappey-Marinier. An automated string-based approach to extracting and characterizing White Matter fiber-bundles. *Computers in Biology and Medicine*, 77:64–75, 2016. Elsevier.

[15] W.W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.*, pages 201–212. ACM Press, 1998.

[16] A.E. Eiben, J.E. Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[17] A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[18] G. Fortino, R. Giannantonio, R. Gravina, P. Kuryloski, and R. Jafari. Enabling effective programming and flexible management of efficient body sensor network applications. *IEEE Transactions on Human-Machine Systems*, 43(1):115–133, 2013.

[19] G. Fortino, A. Guerrieri, G.M.P. O'Hare, and A.G. Ruzzelli. A flexible building management framework based on wireless sensor and actuator networks. *J. Network and Computer Applications*, 35(6):1934–1952, 2012.

[20] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[21] E. Garyfallidis, M. Brett, M.M. Correia, G. Williams, and I. Nimmo-Smith. Quickbundles, a method for tractography simplification. *Front. Neurosci.*, 6:175, 2012.

[22] P. Gawrychowski and P. Uznanski. Order-preserving pattern matching with k mismatches. In *Combinatorial Pattern Matching - 25th Annual Symposium, CPM 2014, Moscow, Russia, June 16-18, 2014. Proceedings*, volume 8486 of *LNCS*, pages 130–139. Springer, 2014.

[23] M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics*. Springer, 2nd edition, 2010.

[24] A. Ghosh and S.K. Kuttal. Semantic clone detection: Can source code comments help? In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2018, Lisbon, Portugal, October 1-4, 2018*, pages 315–317. IEEE Computer Society, 2018.

[25] C.W.J. Granger and R. Joyeux. An introduction to longmemory time series models and fractional differencing. *Journal of Time Series Analysis*, 1(1):15–29, 1980.

[26] L. Gravano, P.G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an RDBMS for web data integration. In *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*, pages 90–101. ACM, 2003.

[27] G. Greco and G. Terracina. Frequency-based similarity for parameterized sequences: Formal framework, algorithms, and applications. *Inf. Sci.*, 237:176–195, 2013.

[28] C. Hazay, M. Lewenstein, and D. Sokol. Approximate parameterized matching. *ACM Trans. Algorithms*, 3(3):29, 2007.

[29] Matthew A Jaro. *UNIMATCH, a Record Linkage System: Users Manual*. Bureau of the Census, 1980.

[30] T. Kamiya, S. Kusumoto, and K. Inoue. Ccfinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Trans. Software Eng.*, 28(7):654–670, 2002.

[31] O. Keller, T. Kopelowitz, and M. Lewenstein. On the longest common parameterized subsequence. *Theor. Comput. Sci.*, 410(51):5347–5353, 2009.

[32] S. Kirkpatrick., C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[33] G.M. Landau and U. Vishkin. Fast parallel and serial approximate string matching. *J. Algorithms*, 10(2):157–169, 1989.

[34] V. I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8):707–710, 1966.

[35] Y. Li, H. Shi, F. Han, Z. Duan, and H. Liu. Smart wind speed forecasting approach using various boosting algorithms, big multi-step forecasting strategy. *Renewable Energy*, 135:540–553, 2019.

[36] J. Mårtensson, M. Nilsson, F. Ståhlberg, P.C. Sundgren, C. Nilsson, D. van Westen, E.M. Larsson, and J. Lätt. Spatial analysis of diffusion tensor tractography statistics along the inferior fronto-occipital fasciculus with application in progressive supranuclear palsy. *MAGMA*, 26(6):527–537, Dec 2013.

[37] J. Mendivelso and Y. J. Pinzón. Parameterized matching: Solutions and extensions. In *Stringology*, pages 118–131. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2015.

[38] X. Miao, H. Song, T. Biming, and P. Sazia. Anomaly detection in wireless sensor networks: A survey. *J. of Network and Computer Applications*, 34(4):1302 – 1325, 2011.

[39] S. Mori, B.J. Crain, V.P. Chacko, and P.C.M. van Zijl. Three-dimensional tracking of axonal projections in the brain by magnetic resonance imaging. *Ann. Neurol.*, 45(2):265–269, 1999.

[40] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[41] N. Potha, M. Maragoudakis, and D. Lyras. A biology-inspired, data mining framework for extracting patterns in sexual cyberbullying data. *Knowl.-Based Syst.*, 96:134–155, 2016.

[42] F. Rodrigues, I. Markou, and F.C. Pereira. Combining time-series and textual data for taxi demand prediction in event areas: A deep learning approach. *Information Fusion*, 49:120–129, 2019.

[43] S. Sargsyan, S. Kurmangaleev, A. Belevantsev, and A. Avetisyan. Scalable and accurate detection of code clones. *Programming and Computer Software*, 42(1):27–33, 2016.

[44] J. Serr and J.Ll. Arcos. An empirical evaluation of similarity measures for time series classification. *Knowl.-Based Syst.*, 67:305–314, 2014.

[45] D. Simon. *Evolutionary optimization algorithms*. John Wiley & Sons, 2013.

[46] T. Smith and M. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[47] C. Stamile, F. Cauteruccio, G. Terracina, D. Ursino, G. Kocevar, and D. Sappey-Marinier. A model-guided string-based approach to white matter fiber-bundles extraction. In *Brain Informatics and Health - 8th International Conference, BIH 2015, London, UK, August 30 - September 2, 2015. Proceedings*, volume 9250 of *LNCS*, pages 135–144. Springer, 2015.

[48] J.M. Vidal, M.A. Sotelo Monge, and L.J. Garca Villalba. A novel pattern recognition system for detecting android malware by analyzing suspicious boot sequences. *Knowl.-Based Syst.*, 150:198–217, 2018.

[49] S. Wagner, G. Kronberger, A. Beham, M. Kommenda, A. Scheibenpflug, E. Pitzer, S. Vonolfen, M. Kofler, S. Winkler, V. Dorfer, and M. Affenzeller. *Advanced Methods and Applications in Computational Intelligence*, volume 6 of *Topics in Intelligent Engineering and Informatics*, chapter Architecture and Design of the HeuristicLab Optimization Environment, pages 197–261. Springer, 2014.

[50] M.S. Waterman, T.F. Smith, and W.A. Beyer. Some biological sequence metrics. *Adv. Math.*, 20(3):367–387, 1976.

[51] M. Wilson, C.R. Tench, P.S. Morgan, and L.D. Blumhardt. Pyramidal tract mapping by diffusion tensor magnetic resonance imaging in multiple sclerosis: improving correlations with disability. *J. Neurol. Neurosurg. Psychiatry*, 74(2):203–207, 2003.

[52] Y. Yao, A. Sharma, L. Golubchik, and R. Govindan. Online anomaly detection for sensor systems: A simple and efficient approach. *Perform. Eval.*, 67(11):1059–1075, 2010.

[53] G.P. Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.

[54] S. Zhang, S. Correia, and D. H. Laidlaw. Identifying white-matter fiber bundles in dti data using an automated proximity-based fiber-clustering method. *IEEE Trans. Vis. Comput. Graph.*, 14(5):1044–1053, Sept 2008.

[55] Y. Zhang and J. Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual reviews in control*, 32(2):229–252, 2008.