On the k-Hamming and k-Edit Distances

Chiara Epifanio[1], Luca Forlizzi[2], Francesca Marzi[2], Filippo Mignosi[2], Giuseppe Placidi[3] and Matteo Spezialetti[2,*]

[1]DMI Department, University of Palermo, Italy
[2]DISIM Department, University of L'Aquila, L'Aquila, Italy
[3]MESVA Department, University of L'Aquila, L'Aquila, Italy

Abstract

In this paper we consider the weighted k-Hamming and k-Edit distances, that are natural generalizations of the classical Hamming and Edit distances. As main results of this paper we prove that for any $k \geq 2$ the DECIS-k-Hamming problem is $\mathbb{P}$-SPACE-complete and the DECIS-k-Edit problem is NEXPTIME-complete. In our formulation, weights are included in the instance description and the cost is not uniform.

Keywords

k-Edit distance, k-Hamming distance, $\mathbb{P}$-SPACE class, NEXPTIME class, Strings Distance Computation

1. Introduction

Measuring how dissimilar two strings are from each other, is a task that occurs often and which has great importance in various practical fields, such as biometric recognition and the study of DNA, up to spell checking. A formal treatment of the problem passes through the definition of a notion of distance between strings. Numerous distance functions have been proposed and studied from a computational point of view in the literature, based on the idea of measuring the minimum number of modification operations, chosen in a given set of admissible operations, necessary to transform one string into another one: two of the best known are certainly the Edit distance and the Hamming distance, but since 1950 other distances have been introduced and scientific studies have been carried on (cf. for instance [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]). String similarity is therefore a classical topic in computer science but still some relevant problems remain open, such as to find a polynomial time algorithm for the edit distance with swaps and non uniform cost on all operations including swaps (cf. [23]). In [19] some partial results on this forty-year open problem are given.

In this paper we consider, among others, an operation that replaces two consecutive characters with other two ones. Clearly this kind of operation includes the swap operation. We will discuss more details on this subject in Section 4.

ICTCS 2023 – 24th Italian Conference on Theoretical Computer Science September 13-15, 2023, Palermo, Italy

*Corresponding author.
chiara.epifanio@unipa.it (C. Epifanio); luca.forlizzi@univaq.it (L. Forlizzi); francesca.marzi@univaq.it (F. Marzi); filippo.mignosi@univaq.it (F. Mignosi); giuseppe.placidi@univaq.it (G. Placidi); matteo.spezialetti@univaq.it (M. Spezialetti)

In this framework, measuring how similar two strings are is then formalized as an optimization problem, i.e. minimizing the amount of operations to transform one into the other one. It is quite useful to also consider the decision version of such problem, in the following way: in any instance there are two words together with a natural number h and we ask whether or not the two given words have a distance (Hamming, edit or another one) that is smaller than or equal to h.

Previous approach could seem well formalized but there is something hidden: is the description (i.e. the cost of each operation) of the distance (Hamming, edit or another one) *included* inside the instances of the problem *or* the description of the distance has to be considered as a constant that can vary depending on the problem but that should not be considered in the asymptotic analysis of the algorithms that solve the problems? Usually the second approach is the one that seems preferred in literature. For instance we say that the complexity of the classical algorithm for the edit distance is $O(nm)$, where n and m are the lengths of the two strings.

On the contrary something different happens in some cases. In [21] it is proved that including the description of a special distance inside the instances gives rise to an $\mathbb{NP}$-hard problem, whilst much later it has been proved that there is a polynomial-time algorithm for the same problem, when the size of the description of the distance is considered as a constant [7, 15]. The interested reader can see [7] and references therein for more details.

In this paper we study the problems of computing the k-Hamming and the k-Edit distances, for $k \geq 2$, in the first setting, i.e. we suppose that the description of the distance, that includes all costs of all operations, is a part of the instances. The study of these problems following the second approach is still open, as discussed in Section 4.

In Section 2 we introduce our notation and some formal definitions. Section 3 is devoted to prove that, for $k \geq 2$, the decision problems of computing the k-Hamming (DECIS-k-Hamming) and the k-Edit (DECIS-k-Edit) are, respectively, $\mathbb{P}$-SPACE-complete and NEXPTIME-complete. To do so, we follow the same strategy for both problems because, in such a way, the proofs are more natural and easier to follow. First we prove the results for $k = 3$, using polynomial time reductions from any $L \in \mathbb{P}$-SPACE to DECIS-3-Hamming and from any $L \in$ NEXPTIME to DECIS-3-Edit, and straightforwardly extend them to larger values of k. Then we reduce (in polynomial time) the problems with $k = 3$ to the respective problems with $k = 2$, proving the results also for these cases. Section 4 concludes the paper foreshadowing possible research developments.

2. Preliminaries

Given a finite alphabet Σ of cardinality σ, a string over Σ is a sequence $w = w_1 w_2 \dots w_n$, with $w_i \in \Sigma$, for any $1 \leq i \leq n$. The number of characters composing a string w is called its *length*, denoted by $|w| = n$. The string of length 0, also called the empty string, is indicated by ϵ. We denote by Σ^* the set of all strings on Σ and by Σ^n the set of all strings of length n in Σ^*. Trivially $\epsilon \in \Sigma^*$, for any Σ. String x is a substring of string w if there exist u and v such that it is possible to write w as the concatenation of u, x and v i.e., such that $w = uxv$. The empty string is a substring of any string.

Given a string v, it is possible to define a set $Op = \{o : \Sigma^* \to \Sigma^*\}$ of operations that allow to modify it in a new string w. Some well-studied subsets of operations are the *edit operations*:

Definition 1. Given a string $v \in \Sigma^*$, we define:

- INSERTION (I, $\epsilon \to a$) allows to insert a character $a \in \Sigma$ in a position i of v, i.e. $w = v_1 \dots v_{i-1} a v_i \dots v_{|v|}$;
- DELETION (D, $a \to \epsilon$) is the removal of character $a = v_i$ in v, i.e. $w = v_1 \dots v_{i-1} v_{i+1} \dots v_{|v|}$;
- SUBSTITUTION (S, $a \to b$) replaces character $a = v_i$ in v with another character $b \in \Sigma$ in the same position, i.e. $w = v_1 \dots v_{i-1} b v_{i+1} \dots v_{|v|}$.

We describe here another operation that allows to define some more distances.

Definition 2. Given a string $v \in \Sigma^*$, we define k-SUBSTITUTIONS as follow: k-SUBSTITUTION (kS, $a_1 \dots a_k \to b_1 \dots b_k$) allows substitutions of k consecutive characters all at once. It replaces in v the substring $a_1 \dots a_k = v_i \dots v_{i+k-1}$ with $b_1 \dots b_k$, i.e. $w = v_1 \dots v_{i-1} b_1 \dots b_k v_{i+k} \dots v_{|v|}$.

Obviously in Definition 2, kS is a generalization of the S operations, e.g. $S = kS$ if $k = 1$. Notice also that the classical swap operation (see [23] for a formal definition) is a special 2-Substitution.

For the sake of readability, we henceforth use the notation $Op = \{A_1, \dots, A_m\}$, with $A_1, \dots, A_m \in \{I, D, kS |\ k \in \mathbb{Z}^+\}$ to indicate that all the possible operations defined by each A_i are in Op, e.g. $Op = \{I\}$ allows all the insertions $\epsilon \to a$ with $a \in \Sigma$.

At this point, we can define a *cost* function $\gamma : Op \to \mathbb{Z}^+$ for each operation. That cost can be constant or non uniform i.e. it can depend on the operation on which it is applied. More general operations, that includes the ones considered in this paper, have been studied in [19, Section 4].

Notice that, from a formal point of view, all the above operations should have as parameters the positions where they are applied, even if their cost does not depend on them. However, we prefer to not be strictly formal in order to improve the readability of the text.

Definition 3. Let $v, w \in \Sigma^*$ be two strings, Op be a set of operations defined on Σ^*, γ be an arbitrary cost function. If $T = t_1 t_2 \dots t_p$ is a sequence of operations over Op, the overall cost of the sequence is:

$$\gamma(T) = \sum_{i=1}^{p} \gamma(t_i).$$

The *distance* between v and w is the minimum cost required to transform v into w through a sequence of operations T in Op, i.e. if $T(v) = t_p(\dots(t_2(t_1(v)))\dots)$

$$\delta(v, w) = \min\{\gamma(T) \mid T(v) = w\}. \tag{1}$$

Depending on the set Op of operations we can define different distances.

Definition 4. The *Edit distance* between v and w is $\delta(v, w)$, considering $Op = \{I, D, S\}$.

The Edit distance is also formally known as *Levenshtein distance*, due to the work carried out by Vladimir Levenshtein who introduced for the first time an algorithmic approach to calculate this distance [14].

Definition 5 ([13]). We define *Hamming distance* between v and w $\delta(v, w)$, when $Op = \{S\}$.

Apart from the well-studied Edit Distance and Hamming Distance, it is possible to define some other distances between strings such as the following ones, that are special cases of the maximal generalization given in [19].

Definition 6 (2-Edit Distance). The 2-*Edit distance* between two strings v and w is the minimum cost to transform the string v into w, $\delta(v, w)$, setting the set of admissible operations $Op = \{I, D, S, 2S\}$.

Obviously, 2-Edit Distance is a direct extension of the previously defined Edit distance, with the addition of the double substitution operation.

Last but not least we define the generalizations of 2-Edit and Hamming distances, the k-Edit and k-Hamming distance, respectively, for an integer $k \geq 2$.

Definition 7 (k-Edit Distance, k-Hamming Distance). Given two strings v and w and an integer $k \geq 2$,

- the k-*Edit distance* between v and w is $\delta(v, w)$, with $Op = \{I, D, S, k\mathrm{S}\}$.
- the k-*Hamming distance* between v and w is $\delta(v, w)$, with $Op = \{k\mathrm{S}\}$.

3. Complexity

3.1. DECIS-3-Hamming $\mathbb{P}$-SPACE completeness

We prove in this section that DECIS-3-Hamming problem is $\mathbb{P}$ - SPACE-complete. DECIS-3-Hamming contains all the strings encoding quadruples of the form $< v, w, D, h >$ where v and w are two strings on Σ^n of the same length n, D is an encoding string that describes the weighted 3-Hamming distance we are considering, h is an integer and $D(v, w) \leq h$.

Hence, an instance $x =< (v, w, D, h) >$ fits into DECIS-3-Hamming if and only if $D(v, w) \leq h$. Therefore

$$\text{DECIS-3-Hamming} = \{< (v, w, D, h) >: D(v, w) \leq h\}.$$

In order to say that DECIS-3-Hamming is $\mathbb{P}$-Space complete, we need to prove the two following properties: a) DECIS-3-Hamming is in $\mathbb{P}$-Space; b) for every language L in $\mathbb{P}$-Space there exists a polynomial reduction from L to DECIS-3-Hamming.

Theorem 1. *DECIS-3-Hamming is in $\mathbb{P}$-Space.*

Proof. By a corollary to Savitch's Theorem [24] we know that $\mathbb{P}$-Space=$\mathbb{NP}$-Space. Hence, proving that the problem is in $\mathbb{NP}$-Space will be enough to prove the Theorem.

We define a Nondeterministic Turing Machine N that accepts the DECIS-3-Hamming language in polynomial space, even in the worst case. N starts with $< v, w, D, h >$ coded on its tape and operates iteratively. In each loop, it non-deterministically chooses a substitution to apply to the string, executes it and updates h by subtracting the weight of the substitution just chosen. N exits the while loop when v becomes equal to w or h is negative. In both cases it will be possible to establish whether the given instance belongs to DECIS-3-Hamming. It is possible to observe that the total occupied space is linear with respect to the length of the input strings, hence DECIS-3-Hamming is in $\mathbb{NP}$-SPACE, and, therefore, in $\mathbb{P}$-SPACE.

```
begin
    while v ≠ w ∧ h ≥ 0
        non-deterministically choose a substitution to apply;
        apply the substitution to string v;
        subtract the weight of the substitution from h;
    if v == w ∧ h ≥ 0
        ACCEPT
    else
        DO NOT ACCEPT
end
```

Algorithm 1: Algorithm followed by N for solving DECIS-3-Hamming

□

Theorem 2. *For each language L in $\mathbb{P}$-SPACE there is a polynomial time reduction from L to DECIS-3-Hamming.*

Proof. If L is in $\mathbb{P}$-SPACE there exists a deterministic Turing machine

$$M =< Q, \Gamma, B, \Sigma, \Delta, q_0, F >$$

that stops on every input of size n in $O(c^{q(n)})$ time and decides L in polynomial space $O(p(n))$, being c a constant and p and q two polynomials.

We define M' as the Turing Machine that accepts the DECIS-3-Hamming language. We define an algorithm for mapping each instance x in L into an instance $x' =< (v, w, D, h) >$, such that M accepts x if and only if M' accepts x'. We formally define the parameters of instance x' as follows.

- $v = \$B^{p(n)+1}q_0xB^{p(n)+1}\$$, where $\$ \notin \Gamma$;
- $w = \$B^l\$$ with $l = 2p(n) + n + 3$;
- $h = \min\{c^m > c^{q(n)} + 2p(n) + 4 + n\}$. This value of h can be represented in base c as the string obtained by the concatenation of 1 and m times 0, with $m = \lceil \log_c c^{q(n)} + 2p(n) + 4 + n \rceil$.

The last parameter to define is the distance D. We note immediately that the description of the distance is independent of x, therefore it is constant with respect to n. This distance is a weighted 3-Hamming that assumes only two weights 1 and $h+1$. To give the full description of D we would need to define the weight for all 3-substitutions. For each $y \in \Gamma$ the following 3-substitutions with cost 1 are produced:

- every transition $\Delta(q_h, a) = (q_j, b, R)$ in M produces $yq_ha \to ybq_j$ in D;
- every transition $\Delta(q_h, a) = (q_j, b, L)$ in M produces $yq_ha \to q_jyb$ in D;
- every transition $\Delta(q_h, B) = (q_j, b, R)$ in M produces $yq_hB \to ybq_j$ in D;
- every transition $\Delta(q_h, B) = (q_j, b, L)$ in M produces $yq_hB \to q_jyb$ in D.

In addition, the following 3-substitutions with cost 1 are added, for each $q_s \in F$ and $a, b \neq \$$, with $\#_l, \#_r \notin \Gamma$.

- $aq_sb \to \#_lB\#_r$
- $a\#_lB \to \#_lBB$
- $\$\#_lB \to \$BB$
- $B\#_rb \to BB\#_r$
- $B\#_r\$ \to BB\$$

This set of 3-substitutions is required if a $q_s \in F$ appears on the simulated tape. In fact, it is used to erase the entire tape. For the remaining undefined 3-substitutions we set the cost to $h+1$. □

It is possible to observe that the algorithm is polynomial.

Theorem 3. *Let x be an instance in $L \in \mathbb{P}$-SPACE, the transformation of x in x' just defined is a reduction, i.e.*

$$x \in L \iff x' \in \textit{ DECIS-3-Hamming}.$$

Proof. Suppose first that $x \in L$. This means that there exists a finite sequence of ID $\alpha_1 \ldots \alpha_t$ such that $\alpha_1 = q_0x$, for any $i < t < c^{q(n)}$ $\alpha_i \vdash \alpha_{i+1}$ and α_t is a final ID. For each implication from one ID to another one there is a corresponding transition rule which can be simulated by a substitution of unit weight in the distance D, as previously described. Formally we match α_1 to the string v and at the end of the simulation we will have reached α_t which will correspond to a string v' containing q_s. In this way we will be able to say that there exists a sequence of substitutions of unitary weight in D which, starting from v, allows us to arrive at v' with a total weight less than $c^{q(n)}$. Using, at this point, the substitutions of unitary weight that cancel the symbols different from \$ and B around q_s we will obtain the string $w = \$B^l\$$, with $l = 2p(n) + n + 3$. In total, therefore, the cost of obtaining w is less than or equal to $c^{q(n)} + 2p(n) + n + 4$ and therefore less than h. So $x' \in$ DECIS-3-Hamming.

Let us prove now the converse. We do it by contraposition. If $x \notin L$ then there is no sequence of transitions that can lead the initial ID to an ID in which a final state appears. In the simulation using 3-substitutions, no sequence of substitutions of unitary weight can ever transform the string v into a string v' containing a final state and therefore w cannot be obtained. The only way to get an accepting state on the tape would be to use a substitution costing $h+1$. But in this case the 3-Hamming distance between v and w will certainly be greater than h, so $x' \notin$ DECIS-3-Hamming. □

Theorem 4. *Any DECIS-k-Hamming, with $k \geq 3$, is $\mathbb{P}$-SPACE-complete.*

Proof. It is easy to observe that the previous proof can be used to demonstrate, by induction, the $\mathbb{P}$-SPACE-completeness of any DECIS-k-Hamming problem, with $k \geq 3$, since: a) Algorithm 1 works for any DECIS-k-Hamming; b) There exists a polynomial time reduction from DECIS-k-Hamming to DECIS-$(k+1)$-Hamming ($k \geq 2$). The reduction has just to pad input and target string (to handle strings with length $k+1$) and to inhibit any $(k+1)$-substitution that does not represent a k-substitution. □

3.2. DECIS-2-Hamming $\mathbb{P}$-Space-Completeness

In this section we prove that also DECIS-2-Hamming is $\mathbb{P}$-Space Complete. We first define the following set, for any $k \in \mathbb{Z}^+$ and $x, y \in \Sigma^k$:

$$\text{DECIS'-}k\text{-Hamming} = \{< v, w, D, h > | \delta(v, w) \leq h, \gamma(x \to y) \in \{1, h+1\}\}$$

Notice that the proof of $\mathbb{P}$-Space-completeness of DECIS-3-Hamming holds for DECIS'-3-Hamming, too. In fact we have that: a) DECIS'-3-Hamming is a special case of DECIS-3-Hamming, thus the Algorithm 1 is valid; b) the reduction defined in Theorem 2 actually produces instances of DECIS'-3-Hamming.

We can, therefore, state the following lemma.

Lemma 1. *DECIS'-3-Hamming is $\mathbb{P}$-Space-Complete.*

It is also worth noting that an algorithm similar to Algorithm 1 can be defined for DECIS-2-Hamming, thus:

Lemma 2. *DECIS-2-Hamming $\in \mathbb{P}$-Space.*

Lemma 3. *There is a reduction from DECIS'-3-Hamming to DECIS-2-Hamming.*

Proof. It is possible to prove this reduction thanks to a technique which belongs to the folklore of Information Theory and to Markov chains. This technique reduces the dependence of a random variable on k previous random variables, including itself, to just two random variables, including itself, via a sliding window over a larger alphabet.

Let $x =< v, w, D, h >$ be an instance in DECIS'-3-Hamming, we transform it into an instance $x' =< v', w', D', h' >$ in DECIS-2-Hamming, where

- $v' = c_1 c_2 \ldots c_{n+1}$ is obtained from $v = a_1 a_2 \ldots a_n$, by:
  - \$-padding v, i.e. $\bar{v} = b_1 b_2 \ldots b_{n+2} = \$v\$$;
  - coding any symbol of v' as a pair of consecutive symbols of $\bar{v}$, obtained with a sliding window of length 2 and stride 1, i.e. $c_i = (b_i, b_{i+1})$

- w' is constructed from w in an analogous way;
- $h' = 3h$;
- for each 3-substitution $abc \rightarrow def$, with $\gamma = 1$ in D, the following unit cost 2-substitutions are added to D':
  - $(ab)(bc) \rightarrow S^{\leftarrow}_{(ab)(de)} S^{\rightarrow}_{(bc)(ef)}$;
  - $(xa)S^{\leftarrow}_{(ab)(de)} \rightarrow (xd)(de), \forall x \in \Sigma \cup \{\$\}$
  - $S^{\rightarrow}_{(bc)(ef)}(cx) \rightarrow (ef)(fx), \forall x \in \Sigma \cup \{\$\}$
- any other 2-substitution has cost $h' + 1$

The algorithm is polynomial in the size of the input, indeed: a) $|v'| = |v|+1$ and $|w'| = |w|+1$; b) coding $h' = 3h$ requires linear time; c) the algorithm increases the size of the alphabet with a polynomial function and coding D' requires $O(|\Sigma'|^2)$ steps.

Moreover, it is possible to observe that the algorithm is a reduction, i.e.:

$$x \in \text{ DECIS'-3-Hamming } \iff x' \in \text{ DECIS-2-Hamming}$$

Suppose $x \in$ DECIS'-3-Hamming, i.e. $\exists T = t_1 \ldots t_k \ s.t. \ T(v) = w, \ \gamma(T) \leq h$, with each $t_i \in D$. Then, $\exists T' = t'_1 \ldots t'_{3k} \ s.t. \ T'(v') = w', \ \gamma(T') \leq h'$, with each $t'_i \in D'$. T' is obtained by T, by translating each t_i into the corresponding sequence of 2-substitutions described by the algorithm, thus

$$x \in \text{ DECIS'-3-Hamming} \Rightarrow x' \in \text{ DECIS-2-Hamming}$$

Suppose $x \notin$ DECIS'-3-Hamming, i.e. $\forall T = t_1 \ldots t_k \ s.t. \ T(v) = w, \ \gamma(T) > h$, with each $t_i \in D$. Since the algorithm, by construction, do not insert any $S^{\leftarrow}_{(ab)(de)}$ or $S^{\rightarrow}_{(bc)(ef)}$ symbols in w', the only way to obtain w' from v' is to remove all these symbols from the string, thus completing simulated (and legal) 3-substitutions in the input instance. Therefore, $\forall T' \ s.t. \ T'(v') = w', \ \gamma(T') > 3h$, thus

$$x \notin \text{ DECIS'-3-Hamming} \Rightarrow x' \notin \text{ DECIS-2-Hamming}$$

□

These lemmas imply the following result.

Theorem 5. *Decis-2-Hamming is $\mathbb{P}$-Space-Complete.*

3.3. DECIS-3-Edit NEXPTIME-completeness

We will now prove that DECIS-3-Edit distance is NEXPTIME-complete, that is: a) DECIS-3-Edit $\in$ NEXPTIME; b) $\forall L \in$ NEXPTIME, there exists a polynomial time reduction from L to DECIS-3-Edit.

Theorem 6. *DECIS-3-Edit $\in$ NEXPTIME*

Proof. We show a Nondeterministic Turing Machine N that, given $x =< (v, w, D, h) >$ in input, accepts if and only if $D(v, w) < h$. N acts as described in Algorithm 2.

begin
    while $v \neq w \wedge h \geq 0$
        non-deterministically choose an edit operation o to apply;
        apply o to string v;
        subtract $\gamma(o)$ from h;
    if $v == w \wedge h \geq 0$
        ACCEPT
    else
        DO NOT ACCEPT
end

Algorithm 2: Algorithm followed by N for solving DECIS-3-Edit

Since $\gamma : \Sigma^* \times \Sigma^* \to \mathbb{Z}^+$, the algorithm performs at most $h = O(2^n)$ loops, each composed by linear time operations. Thus, M halts in an exponential time in n and DECIS-3-Edit $\in$ NEXPTIME. $\square$

Theorem 7. *$\forall L \in$ NEXPTIME, there exists a polynomial time reduction from L to DECIS-3-Edit*

Proof. If $L \in$ NEXPTIME, there exists a Nondeterministic Turing Machine

$$N' =< Q, \Gamma, B, \Sigma, \Delta, q_0, F >$$

that recognizes if $x \in L$ and stops within an exponential number of moves, i.e. if $n = |x|$, it will halt after $2^{p(n)}$ steps at most, where $p(n)$ is a polynomial function of n. The reduction transforms any instance x for L in an instance $x' =< (v, w, D, h) >$ for DECIS-3-Edit as follows:

- $v = \$q_0x\$$, with $\$ \notin \Gamma$
- $w = \$\$$
- $h = 5 * 2^{p(n)} + 2 * (n + 1)$

Finally, D is defined in the following way:

1. any insertion has cost $\gamma = h + 1$, with the exception of $\epsilon \to B_1$ (being $B_1 \notin \Gamma$ a new blank symbol), that has cost $\gamma = 1$;
2. any deletion has cost $\gamma = h + 1$, with the exception of $* \to \epsilon$, that costs $\gamma = 1$, where $* \notin \Gamma$ is a new symbol used to delete the simulated tape after the acceptance of N';
3. any substitution has cost $\gamma = h + 1$
4. any 3-substitution has cost $\gamma = h + 1$, with the following exceptions:
    a) for each element of $\{< (q, a), (p, b, R) > | (p, b, R) \in \Delta(q, a)\}$, with q and p state symbols not in Γ and $a, b \in \Gamma$:
        i. $qax \to bpx$, with $\forall x \in \Gamma$, has cost $\gamma = 3$;

ii. $qa\$ \rightarrow bp\$$, with $\forall p \notin F$, has cost $\gamma = 1$;
iii. $qa\$ \rightarrow bp\$$, with $\forall p \in F$, has cost $\gamma = 3$;

b) for each element of $\{< (q,a),(p,b,L) > | (p,b,L) \in \Delta(q,a)\}$, with q and p state symbols not in Γ and $a, b \in \Gamma$:
   i. $xqa \rightarrow pxb$, with $\forall x \in \Gamma$, has cost $\gamma = 3$;
   ii. $\$qa \rightarrow p\$b$, with $\forall p \in Q$, has cost $\gamma = 1$;

c) to simulate moves that require to expand the tape length behind $|x|$, the following 3-substitutions have cost $\gamma = 1$:
   i. $qB_1\$ \rightarrow qB\$$;
   ii. $p\$B_1 \rightarrow \$pB$;

d) to delete symbols and reach the target string \$\$ after the acceptance N', with $p \in F$ and $a, b \in \Gamma$, the following 3-substitutions have cost $\gamma = 1$:
   i. $apb \rightarrow \#_l * \#_r$;
   ii. $\$pa \rightarrow \$ * \#_r$;
   iii. $ap\$ \rightarrow \#_l * \$$;
   iv. $\$p\$ \rightarrow \$ * \$$;
   v. $a\#_l* \rightarrow \#_l * *$;
   vi. $*\#_r a \rightarrow * * \#_r$;
   vii. $\$\#_l* \rightarrow \$ * *$;
   viii. $*\#_r\$ \rightarrow * * \$$.

The algorithm takes polynomial time $q(n)$: writing v and w requires linear time in n, while coding D would take $O(|\Gamma|^6)$. Moreover, it is actually a reduction, i.e.:

$$x \in L \iff x' \in \text{ DECIS-3-Edit.}$$

Suppose $x \in L$. There exists a finite sequence of non-deterministic moves (and, therefore, of IDs) that makes N' accept x. It is easy to see that there is a corresponding sequence of transformations that modifies v and results in the string $\$xpy\$$, with $x, y \in \Gamma^*$ and $p \in F$. Each 3-substitution that simulates a N' move has cost $\gamma = 3$, if it does not involve the \$ symbol (or if it ends in a final state symbol within the \$ symbols), otherwise it has cost $\gamma = 1$ if it results in one of the strings: $\$xp\$$ ($p \notin F$, $x \in \Gamma^*$), $p\$x\$$ ($x \in \Gamma^*$). In the latter cases γ has a reduced value because the insertion of B_1 (point 1) and a further 3-substitution are needed to obtain a string that correctly represents the output ID. In any case, a move of N' is simulated by a sequence of transformation S, such that $\gamma(S) = 3$, and, therefore, a sequence of moves from the initial ID to an accepting one can be simulated with a total cost $3 * 2^{p(n)}$. At this point, a sequence of 3-substitutions has to be applied to transform all the symbols within the two \$ into $*$. They are $n + 1 + 2^{p(n)}$ at most and each 3-substitution adds one $*$ at unitary cost. Thus, the whole sequence has cost $\gamma \leq n + 1 + 2^{p(n)}$. Finally, the same cost is required by the sequence of deletions that results in the string \$\$. Thus,

$$x \in L \Rightarrow \delta(\$q_0x\$, \$\$) \leq h.$$

Suppose now $x \notin L$. Any 3-substitution that does not correspond to a legal move of N', or is part of it, has cost $h + 1$, with the exception of those used to transform symbols into $*$

and they can be applied only when the simulated ID is an accepting one. The same holds for deletion of $*$ symbols. Thus, all the sequences of transformations from $\$q_0x\$$ to $\$\$$ have cost $\gamma > n + 1 + 2^{p(n)}$, i.e.:

$$x \notin L \Rightarrow \delta(\$q_0x\$, \$\$) > h$$

□

Theorem 8. *Any DECIS-k-Edit, with $k \geq 3$, is NEXPTIME-complete.*

Proof. The proof is analogue to that of Theorem 4. It is easy to demonstrate, by induction, the NEXPTIME-completeness of any DECIS-k-Edit problem, with $k \geq 3$, since: a) Algorithm 2 works for any DECIS-k-Edit; b) There exists a polynomial time reduction from DECIS-k-Edit to DECIS-$(k+1)$-Edit ($k \geq 2$). Again, the reduction has to inhibit any $(k+1)$-substitution not representing a k-substitution. □

3.4. DECIS-2-Edit NEXPTIME-completeness

We now prove the NEXPTIME-completeness of DECIS-2-Edit. To prove that DECIS-2-Edit $\in$ NEXPTIME, one can employ the same algorithm given in Section 3.3 (Algorithm 2). Instead of explicitly showing that exists a polynomial time reduction from any problem in NEXPTIME to DECIS-2-Edit, we show a polynomial time reduction from a NEXPTIME-complete problem. Indeed, we proved in Section 3.3 the NEXPTIME-completeness of DECIS-3-Edit, but the same proof actually holds for a restricted version of the problem, named DECIS'-3-Edit, where for each instance $< v, w, D, h >$: a) any insertion, deletion or substitution costs either 1 or $h+1$; b) 3-substitutions costs are limited to 1, 3 or $h+1$.

Therefore, to prove the NEXPTIME-completeness of DECIS-2-Edit, it is sufficient to show a reduction from DECIS'-3-Edit to it.

Theorem 9. *There exists a polynomial time reduction from DECIS'-3-Edit to DECIS-2-Edit.*

Proof. The reduction transforms any instance $x =< (v, w, D, h) >$ for DECIS'-3-Edit in an instance $x' =< (v, w, D', 5h) >$ for DECIS-2-Edit as follows:

1. if Σ is the alphabet of the input instance, Σ' for the output instance is augmented by adding the following new symbols:
   a) $S^i_{(abc)(def)}, \forall a, b, c, d, e, f \in \Sigma, i \in \{1, 2, 3\}$;
   b) the supporting symbol $*$;
2. for each $\epsilon \to a \in D$ s.t. $\gamma(\epsilon \to a) = 1$, add $\epsilon \to a$, with $\gamma = 5$, in D';
3. for each $a \to \epsilon \in D$, s.t. $\gamma(a \to \epsilon) = 1$, add $a \to \epsilon$, with $\gamma = 5$, in D';
4. for each $a \to b \in D$ s.t. $\gamma(a \to b) = 1$, add $a \to b$, with $\gamma = 5$, in D'
5. for each 3-substitution $abc \to def \in D$ s.t. $\gamma(abc \to def) = k \leq h$, add the following operations to D':
   a) $\epsilon \to S^1_{(abc)(def)}$, with $\gamma = 5k - 4$;
   b) $aS^1_{(abc)(def)} \to dS^2_{(abc)(def)}$, with $\gamma = 1$;
   c) $S^2_{(abc)(def)}b \to eS^3_{(abc)(def)}$, with $\gamma = 1$;

d) $S^3_{(abc)(def)}c \to f*$, with $\gamma = 1$;
e) $* \to \epsilon$, with $\gamma = 1$;

6. any other operation has cost $5h + 1$ in D'.

The algorithm requires polynomial time. Source and target strings are unchanged, the limit h has to be multiplied by 5 and the size of the alphabet (and of D') is increased by a polynomial function: $|\Sigma'| = O(|\Sigma|^6)$.

We can observe that the algorithm is actually a reduction, i.e:

$$x \in \text{ DECIS'-3-Edit} \iff x' \in \text{ DECIS-2-Edit}$$

Suppose $x =< v, w, D, h >\in$ DECIS'-3-Edit, i.e. $\exists T\ s.t.\ T(v) = w, \gamma(T) \leq h$. Let $T = t_1 t_2 \dots t_n$: it is possible to "simulate" each t_i on x' with a sequence of one or more operations T_i' at cost $\gamma(T_i') = 5 * \gamma(t_i)$. Insertions, deletions and substitutions require a single operation, while, for 3-substitutions, the whole sequence of operations described at point 5 is needed, with total cost of $5k$, where k is the original 3-substitution cost. Therefore,

$$x =< v, w, D, h >\in \text{ DECIS'-3-Edit} \Rightarrow x' =< v, w, D', 5h >\in \text{ DECIS-2-Edit}$$

On the other hand, suppose $x =< v, w, D, h >\notin$ DECIS'-3-Edit. It can be observed that each operation on x' either:

- has cost larger than $5h + 1$ and can not be part of an acceptable sequence;
- corresponds to an operation t_i on x, with cost $5 * \gamma(t_i)$;
- is part of a 3-substitution simulation. Each step of the sequence can be executed only after the previous and the first step introduces a symbol that can not be part of w'. The only possibility to remove "exogenous" symbols is to apply all the operations in the sequence, at cost $5 * \gamma(t_i)$, where t_i is the simulated 3-substitution.

Therefore,

$$x =< v, w, D, h >\notin \text{DECIS'-3-Edit} \Rightarrow x' =< v, w, D', 5h >\notin \text{DECIS-2-Edit}$$

□

4. Conclusions

In this work we studied the computational complexity of the problems of computing the cost of the k-Hamming and k-Edit distances, for $k \geq 2$, proving that the decision versions that include the description of the distance as part of the instances are, respectively, $\mathbb{P}$-SPACE-complete and NEXPTIME-complete. Negative results as these ones are of theoretical relevance but can also facilitate further researches, as discussed in the following.

We have some preliminary results, not included in this paper, for some special cases where the size of the description of the distance is considered constant. For instance, we found a polynomial time algorithm to compute the 2-Hamming distance when every operation has the same constant cost.

It is an open problem to find the complexity of solving both problems as the lengths of the two words increase when the distance is fixed, or, more generally, when the complexity is further parameterized analogously as done in [7, 15] for the swap-insert correction distance.

This new open problem is thus connected with the forty-year open problem contained in [23], since swap operations are special 2-Substitutions.

Our results suggest that if polynomial algorithms exist, they must non-polinomially depend on some parameter such as the maximum of the ratio between all possible operations costs.

Acknowledgments

This research is funded by FONDO FINALIZZATO ALLA RICERCA DI ATENEO (FFR), University of Palermo, year 2023.

References

[1] A. Amir, E. Eisenberg, E. Porat, Swap and mismatch edit distance, Algorithmica 45 (2006) 109–120.

[2] M. Anselmo, G. Castiglione, M. Flores, D. Giammarresi, M. Madonia, S. Mantaci, Isometric words based on swap and mismatch distance, in: Developments in Language Theory. DLT23, volume 13911 of *Lect. Notes Comput. Sci.*, Springer Nature Switzerland, 2023, pp. 23–35.

[3] M. Anselmo, G. Castiglione, M. Flores, D. Giammarresi, M. Madonia, S. Mantaci, Hypercubes and isometric words based on swap and mismatch distance, in: Descriptional Complexity of Formal Systems. DCFS 2023, volume 13918 of *Lect. Notes Comput. Sci.*, Springer, 2023, pp. 21–35.

[4] M. Anselmo, M. Flores, M. Madonia, Quaternary n-cubes and isometric words, in: T. Lecroq, S. Puzynina (Eds.), Combinatorics on Words, volume 12842 of *Lect. Notes Comput. Sci.*, Springer International Publishing, 2021, pp. 27–39.

[5] M. Anselmo, M. Flores, M. Madonia, On k-ary n-cubes and isometric words, Theor. Comput. Sci. 938 (2022) 50–64.

[6] A. Backurs, P. Indyk, Edit distance cannot be computed in strongly subquadratic time (unless SETH is false), SIAM J. Comput. 47 (2018) 1087–1097. URL: https://doi.org/10.1137/15M1053128.

[7] J. Barbay, P. Pérez-Lantero, Adaptive computation of the swap-insert correction distance, ACM Trans. Algorithms 14 (2018). URL: https://doi.org/10.1145/3232057.

[8] M.-P. Béal, M. Crochemore, Checking whether a word is hamming-isometric in linear time, Theoretical Computer Science 933 (2022) 55–59.

[9] L. Bergroth, H. Hakonen, T. Raita, A survey of longest common subsequence algorithms, in: Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00), SPIRE '00, IEEE Computer Society, USA, 2000, p. 39.

[10] G. Cormode, S. Muthukrishnan, The string edit distance matching problem with moves, ACM Trans. Algorithms 3 (2007). URL: https://doi.org/10.1145/1186810.1186812.

[11] F. J. Damerau, A technique for computer detection and correction of spelling errors, Communications of the ACM 7 (1964) 171–176.

[12] S. Faro, A. Pavone, K. Steinhofel, An efficient skip-search approach to swap matching, The Computer Journal 61 (2018) 1351–1360.

[13] R. W. Hamming, Error detecting and error correcting codes, The Bell system technical journal 29 (1950) 147–160.

[14] V. I. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions and Reversals, Soviet Physics Doklady 10 (1966) 707–710.

[15] D. Meister, Using swaps and deletes to make strings match, Theoretical Computer Science 562 (2015) 606–620.

[16] S. C. Sahinalp, Edit distance under block operations, in: M.-Y. Kao (Ed.), Encyclopedia of Algorithms, Springer US, Boston, MA, 2008, pp. 265–267.

[17] D. Shapira, J. A. Storer, Edit distance with move operations, Journal of Discrete Algorithms 5 (2007) 380–392. 2004 Symposium on String Processing and Information Retrieval.

[18] D. Shapira, J. A. Storer, Edit distance with block deletions, Algorithms 4 (2011) 40–60. doi:10.3390/a4010040.

[19] E. Ukkonen, Algorithms for approximate string matching, Information and Control 64 (1985) 100–118. doi:10.1016/S0019-9958(85)80046-2, International Conference on Foundations of Computation Theory.

[20] R. A. Wagner, On the complexity of the extended string-to-string correction problem, in: Proceedings of the seventh annual ACM symposium on theory of computing, 1975, pp. 218–223.

[21] R. A. Wagner, M. J. Fischer, The string-to-string correction problem, Journal of the ACM (JACM) 21 (1974) 168–173.

[22] R. A. Wagner, R. Lowrance, An extension of the string-to-string correction problem, Journal of the ACM (JACM) 22 (1975) 177–183.

[23] R. A. Wagner, On the complexity of the extended string-to-string correction problem, in: D. Sankoff, J. B. Kruskal (Eds.), Time Warps, String Edits, and Macromolecules The Theory and Practice of Sequence Comparison, Addison-Wesley Publishing Company, 1983, pp. 218–223.

[24] W. J. Savitch, Relationships between nondeterministic and deterministic tape complexities, Journal of Computer and System Sciences 4 (1970) 177–192.