The Normalized Edit Distance with Uniform Operation Costs Is a Metric

Dana Fisman
Dept. of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Joshua Grogin
Dept. of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Oded Margalit
Dept. of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Gera Weiss
Dept. of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Abstract

We prove that the normalized edit distance proposed in [Marzal and Vidal 1993] is a metric when the cost of all the edit operations are the same. This closes a long standing gap in the literature where several authors noted that this distance does not satisfy the triangle inequality in the general case, and that it was not known whether it is satisfied in the uniform case – where all the edit costs are equal. We compare this metric to two normalized metrics proposed as alternatives in the literature, when people thought that Marzal's and Vidal's distance is not a metric, and identify key properties that explain why the original distance, now known to also be a metric, is better for some applications. Our examination is from a point of view of formal verification, but the properties and their significance are stated in an application agnostic way.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases edit distance, normalized distance, triangle inequality, metric

Digital Object Identifier 10.4230/LIPIcs.CPM.2022.17

Related Version *Previous Version*: `https://arxiv.org/abs/2201.06115`

Funding This work was supported in part by ISF grants 2714/19 and 2507/21.

1 Introduction

The *edit distance* [5], also called *Levenshtein distance*, is the minimal number of insertions, deletions or substitutions of characters needed to edit one word into another. This is a commonly used measure of the distance between strings. It is used in error correction, pattern recognition, computational biology, and other fields where the data is represented by strings.

One limitation of the edit distance is that it does not contain a normalization with respect to the lengths of the compared strings. This limits its use because, in many applications, having many edit operations when comparing short strings is more significant than having the same number of edit operations in a comparison of longer strings, i.e., some applications require a measure that captures the "average" number of operations per letter, in some sort.

There are several approaches in the literature to add a normalization factor to the edit distance, as follows. The simplest idea that comes to mind is, of course, to divide the edit distance by the sum of lengths of the strings. However, Vidal and Marzal [8] showed that this function, termed *post-normalized edit distance* in [8], does not satisfy the triangle inequality, and thus is not a metric. Dividing by the length of the minimal or maximal among the strings also breaks the triangle inequality [2]. The fact that a distance measure is (or is not) a metric allows (resp. prevents) optimizations in many applications. For example, many efficient algorithms for searching shortest paths in graphs, such as Dijkstra's algorithm, make use of the fact that the underlying distance is a metric.

Vidal and Marzal propose thus another function, that we will focus on in this paper, that they term *the normalized edit distance* (NED) and say that this function, *"seems more likely to fulfill the triangle inequality"*. They however, show that when the sum of the costs of deleting and inserting a particular symbol is much smaller than any other elemental edit cost the function that they suggest is also non-triangular. The question of whether this distance is triangular in less contrived situations is given only an empirical answer – *"triangular behavior has actually been observed in practice for the normalized edit distance"*. This state of affairs opened the way for attempts to define edit distance functions that are normalized and satisfy the triangle inequality, as discussed in the following two paragraphs.[1]

Li and Liu [6] proposed an alternative normalization method. They open their paper by saying that *"Although a number of normalized edit distances presented so far may offer good performance in some applications, none of them can be regarded as a genuine metric between strings because they do not satisfy the triangle inequality"*. They, then, define a new distance, *the generalized edit distance* (GED), that is a simple function of the lengths of the compared strings and the edit distance between them and show that it is a metric.

De la Higuera and Mičo [2] propose the *contextual normalised edit distance* (CED). Their normalization goes by dividing each edit operation locally by the length of the string on which it is applied. Specifically, instead of dividing the total edit costs by the length of the edit path, they propose to divide the cost of each edit operation by the length of the string at the time of edit. They prove that this is a metric, provide an efficient approximation procedure for it, and demonstrate its performance in several application domains.

In this paper we prove that NED, the original edit normalization approach proposed by Vidal and Marzal [8] does satisfy the triangle inequality when the cost of all the edit operations are the same. Since this setup is very common in many applications of the edit distance, our result gives a simple normalization technique that satisfies the triangle inequality. While there are other normalized edit distance functions that are a metric, in particular the two mentioned above (GED and CED), their definition is more complicated and they capture a different notion of distance than that of NED.

The motivation that led us to engage in distances between words came from the field of formal methods; specifically, for software verification. In this field, it is customary to represent runs of a system using words and analyze the relationship between the set of words that satisfy a given specification and the set of words that the system under examination produces. Naturally, the main question asked is whether there is a word that the system produces that does not satisfy the requirement, but an appropriate concept of distance opens up the possibility of asking further questions. For example, for systems that meet the specifications, the robustness question would be, "is there a run that is closer than a given threshold to not meeting the requirements?". In this context, we would like the distance to measure how much "disturbance" in a word we can afford without risking non-compliance. Naturally, since editing model symmetric disturbances, we use uniform weights. As we will explain in Subsection 3.2 below, the NED distance satisfies certain properties required for use in formal the field of formal methods that other metrics do not. Another advantage of NED in the context of formal methods is that its definition allows direct use of a PTIME algorithm proposed by Filliot et al. [3] for computing the distance between regular sets of

[1] The complexity of computing NED was first shown to be $O(mn^2)$ with experimental data that suggested that it is actually $O(mn)$ [9]. It was later proven to be $O(mn \log n)$ in the uniform case [1]. Here, $n \geq m$ are the lengths of the compared words.

words represented using finite automata. This is useful since verification tools work with automata to represent the specification and the program runs, and verification questions are usually reduced to questions on automata.

2 Preliminaries

Let Σ be a finite alphabet and Σ^* the set of all finite strings over Σ. The length of string $w = \sigma_1\sigma_2\ldots\sigma_n$, denoted $|w|$, is n. We use $w[i]$ for the i-th letter of w, and $w[i..]$ for the suffix of w starting at i, namely $w[i..] = \sigma_i\sigma_{i+1}\ldots\sigma_n$.

Basic and extended edit letters. The literature on defining distance between words over Σ uses the notion of *edit paths*, which are strings over *edit letters* defining how to transform a given string s_1 to another string s_2. The standard operations are *deleting* a letter, *inserting* a letter, or *swapping* one letter with another letter. Formally, the *basic edit letters alphabet* Γ is defined as $\Gamma = \{\mathtt{n}, \mathtt{c}, \mathtt{v}, \mathtt{x}\}$ where:

- `c` stands for *change*: the relevant letter in the source string is replaced with another letter.
- `v` stands for *insert*: a new letter is added to the destination string.
- `x` stands for *delete*: the current letter from the source string is deleted and not copied to the destination string.
- `n` stands for *no-change*: the current letter is copied as is from the source string to the destination string.

The edit letters in Γ do not carry enough information to transform a string w over Σ to an unknown string over Σ, since for instance the letter `v` does not provide information on which letter $\sigma \in \Sigma$ should be inserted. To this aim we define the alphabet Γ_Σ that provides all the information required. Formally, $\Gamma_\Sigma = \{\mathtt{l}_\sigma |\ \sigma \in \Sigma, \mathtt{l} \in \{\mathtt{n}, \mathtt{v}, \mathtt{x}\}\} \cup \{\mathtt{c}_{(\sigma_1,\sigma_2)} |\ \sigma_1, \sigma_2 \in \Sigma\}$. We call strings over Γ_Σ *edit paths*. Throughout this document we use $w, w_1, w_2, w', \ldots$ and $s, s_1, s_2, s', \ldots$ for strings over Σ and $p, p_1, p_2, p', \ldots$ for edit paths.

Weights and length of edit paths. Given a function $wgt\colon \Gamma_\Sigma \to \mathbb{N}$, that defines a weight to each edit letter, we define the weight of an edit path $wgt\colon \Gamma_\Sigma^* \to \mathbb{N}$ as the sum of weights of the letter it is composed from, namely for an edit path $p = \gamma_1\gamma_2\ldots\gamma_m \in \Gamma_\Sigma^*$, $wgt(\gamma_1\ldots\gamma_m) = \sum_{i=1}^{m} wgt(\gamma_i)$.

In our case we are interested in *uniform* costs where the weight of `n` is 0 and the weight of all other operations is the same. For simplicity we can assume that the weight of all other operations is 1. Thus, we can define the weight over Γ instead of Γ_Σ simply as $wgt\colon \Gamma \to \mathbb{N}$ where $wgt(\gamma) = 0$ if $\gamma = \mathtt{n}$ and $wgt(\gamma) = 1$ otherwise, namely if $\gamma \in \{\mathtt{c}, \mathtt{v}, \mathtt{x}\}$. We also define the function $len\colon \Gamma_\Sigma \to \mathbb{N}$ as $len(\gamma) = 1$ and $len\colon \Gamma_\Sigma^* \to \mathbb{N}$ as $len(\gamma_1\ldots\gamma_m) = \sum_{i=1}^{m} len(\gamma_i)$. Clearly here we have $len(p) = |p|$. Later on we will introduce new edit letters whose length is different from 1, thus the need for a definition of *len* that is not just the count of letters.

▶ **Example 1.** Let $w_1 = abcd$ and $w_2 = badee$. Then $p = \mathtt{x}_a \cdot \mathtt{n}_b \cdot \mathtt{c}_{c,a} \cdot \mathtt{n}_d \cdot \mathtt{v}_e \cdot \mathtt{v}_e$ is an edit path transforming w_1 to w_2. We have that $wgt(p) = wgt(\mathtt{xncnvv}) = 4$ and $len(p) = 6$.

Applying an edit path to a string. Given a string w over Σ, and an edit path p over Γ_Σ we can now define the result of applying p to w.

▶ **Definition 2.** *We define a function apply*: $\Sigma^* \times \Gamma_\Sigma^* \to (\Sigma \cup \{\bot\})^*$ *that given a string* w *over* Σ*, and an edit path* p *over* Γ_Σ *returns a new string* w' *over* $\Sigma \cup \{\bot\}$*. If* p *is a valid edit path for* w *it returns a string over* Σ*, otherwise a string that contains* $\bot$*.*

$$apply(p, w) = \begin{cases} \varepsilon & \text{if } p = w = \varepsilon \\ \sigma' \cdot apply(p[2..], w) & \text{if } p[1]{=}\mathtt{v}_{\sigma'} \\ \sigma' \cdot apply(p[2..], w[2..]) & \text{if } p[1]{=}\mathtt{c}_{(\sigma,\sigma')} \text{ and } w[1]{=}\sigma \\ \sigma \cdot apply(p[2..], w[2..]) & \text{if } p[1]{=}\mathtt{n}_{\sigma} \text{ and } w[1]{=}\sigma \\ apply(p[2..], w[2..]) & \text{if } p[1]{=}\mathtt{x}_{\sigma} \text{ and } w[1]{=}\sigma \\ \bot & \text{otherwise} \end{cases}$$

We say that a string p_{ij} over Γ_Σ is an *edit path from* string s_i to string s_j over Σ if $apply(p_{ij}, s_i) = s_j$. With a bit of overriding, we say that a string p_{ij} over Γ is an *edit path from* strings s_i to s_j over Σ if there exists an extension of p_{ij} with subscripts from Σ that results in an edit path from s_i to s_j.

▶ **Example 3.** Following on Ex. 1, we have that $apply(\mathtt{x}_a\mathtt{n}_b\mathtt{c}_{c,a}\mathtt{n}_d\mathtt{v}_e\mathtt{v}_e, abcd) = badee$, and that `xncnvv` is an edit path from *abcd* to *badee*.

The normalized edit distance. Let p be an edit path. The *cost* of p, denoted $cost(p)$ is defined to be the weight of p divided by the length of p, if the length is not zero, and zero otherwise. That is, $cost(p) = 0$ if $|p| = 0$ and $cost(p) = \frac{wgt(p)}{len(p)}$ otherwise.

Using the definition of *cost* we can define the notion we study in this paper, namely the *normalized edit distance*, NED, of Marzal and Vidal [8].

▶ **Definition 4** (The normalized edit distance, NED [8])**.** *The* normalized edit distance *between* s_i *and* s_j*, denoted* $\text{NED}(s_i, s_j)$ *is the minimal cost of an edit path* p_{ij} *from* s_i *to* s_j*. That is,*

$$\text{NED}(s_i, s_j) = \min\{\, cost(p_{ij}) \mid p_{ij} \in \Gamma_\Sigma^* \text{ and } apply(p_{ij}, s_i) = s_j \}$$

Note that while, in general, *wgt* may assign arbitrary weights to edit letters, in this paper we assume the uniform weights as defined above.

▶ **Example 5.** Let $\Sigma = \{a, b, c\}$, $s_1 = acbb$ and $s_2 = cc$. Then the string `xnxc` denotes an edit path taking s_1, deleting the first letter (a), copying the second letter (c), deleting the third letter (b), and replacing the fourth letter (b) by c. This edit path indeed transforms s_1 to s_2. Its cost is $\frac{1+0+1+1}{4} = \frac{3}{4}$. It is not hard to verify that this cost is minimal, therefore $\text{NED}(s_1, s_2) = \frac{3}{4}$.

The alignment view. Recall that distance functions defined by dividing the weight by the sum, max or min of the given strings does not yield a metric [2, 8]. The main contribution of the paper is to show that the choice to use the length of the edit path in the denominator, makes the resulting definition, NED, a metric. To understand the motivation behind dividing by the length of the edit path, note that an edit path can be thought of as defining an alignment between the given words s_1 and s_2 by padding the first string with some blank symbol, denote it `_`, whenever an insert operation is conducted, and padding the second string with `_` symbols whenever a delete operation is conducted. The resulting words s_1' and s_2' would thus be of the same length, and the weight of the edit path would correspond to the Hamming distance between the words. (The Hamming distance applies only to words of same length and counts the number of positions i in which the two words differ.) When dealing with words of the same length it makes sense to normalize them by dividing by their length, and the length of the padded words equals the length of the edit paths.

▶ **Example 6.** In Ex. 5 we used $s_1 = acbb$, $s_2 = cc$. The edit path `xnxc` corresponds to the alignment $s'_1 = acbb$ and $s'_2 = \_c\_c$, and since the length of s'_1 and s'_2 is 4 and they differ in all positions but one the corresponding cost is 3/4.

In Ex. 1, we used $w_1 = abcd$ and $w_2 = badee$ and considered the edit path `xncnvv`. This path correspond to the alignment $w'_1 = abcd\_\_$ and $w'_2 = \_badee$. Since w'_1 and w'_2 differ in four out of the six positions, we have that the cost of this path is 4/6.

A metric space. A metric space is an ordered pair $(\mathbb{M}, d)$ where $\mathbb{M}$ is a set and $d\colon \mathbb{M}\times\mathbb{M} \to \mathbb{R}$ is a *metric*, i.e., it satisfies the following for all $m_1, m_2, m_3 \in \mathbb{M}$:

1. $d(m_1, m_2) = 0$ iff $m_1 = m_2$;
2. $d(m_1, m_2) = d(m_2, m_1)$;
3. $d(m_1, m_3) \leq d(m_1, m_2) + d(m_2, m_3)$.

The first condition is referred to as *identity of indiscernibles*, the second as *symmetry*, the third as the *triangle inequality*.

Basic properties of NED. It is not hard to see that NED satisfies the first and second condition of being a metric. The following proposition establishes that the distance of a string to itself, according to NED, is zero, and that the distance between two strings is symmetric.

▶ **Proposition 7.** *Let* $s, s_1, s_2 \in \Sigma^*$. *Then*

1. $NED(s, s) = 0$
2. *if* $s_1 \neq s_2$ *then* $NED(s_1, s_2) > 0$
3. $NED(s_1, s_2) = NED(s_2, s_1)$

Its straight forward proof can be found in the archived version [4].

The challenge is proving that NED satisfies the third condition, the triangle inequality. We do this in Section 4. Before that we investigate some properties of NED and other edit distance functions.

3 Properties of the various normalized edit distance functions

3.1 Other edit distance functions

In the introduction we mentioned several edit distance functions known to be a metric. We use the term *edit distance* for functions between words to values that are based on *delete*, *insert* and *swaps*. In general these definition may allow arbitrary weight assignment to edit letters, but we consider the case of uniform weights. We start by introducing the edit distance functions, ED, GED, and CED, and then turn to compare their properties, with those of NED.

We start with the commonly used *edit distance*, introduced by Levenstein [5].

▶ **Definition 8** (The edit (Levenstein) distance, ED)**.** *The* edit distance *between* s_i *and* s_j, *denoted* $ED(s_i, s_j)$, *is the minimal weight of a path* p_{ij} *from* s_i *to* s_j. *That is,*

$$ED(s_i, s_j) = \min\{wgt(p_{ij}) \mid p_{ij} \in \Gamma^*_\Sigma \text{ and } apply(p_{ij}, s_i) = s_j\}$$

This function is a metric, but it completely ignores the lengths of the words, thus it is not normalized.

We turn to introduce the *generalized normalized edit distance* proposed and proven to be a metric by Li and Liu [6].

▶ **Definition 9** (The generalized edit distance)**.** $GED(s_i, s_j) = \frac{2\cdot ED(s_i,s_j)}{|s_i|+|s_j|+ED(s_i,s_j)}$.

Last, we define of the *contextual edit distance*, proposed and proven to be a metric by de la Higuera and Micó [2]. It starts with a definition of distance between two strings whose Levenstein distance is 1, from which it builds the distance for an arbitrary set of words, by looking at a sequence of intermediate transformations.

▶ **Definition 10** (The contextual edit distance). *Let s, s' be such that $\text{ED}(s, s') = 1$ their contextual edit distance is defined by $\text{CED}(s, s') = \frac{1}{\max(|s|,|s'|)}$. Note that given $\text{ED}(s, s') = 1$ the difference between the lengths of s and s' is at most one, thus $\max(|s|, |s'|) \leq \min(|s|, |s'|) + 1$.*

Given a sequence of strings $\alpha = (s_0, s_1, \ldots, s_k)$ such that $\text{ED}(s_i, s_{i+1}) = 1$ for all $0 \leq i < k$, one can define $\text{CED}(\alpha) = \sum_{i=1}^{k} \text{CED}(s_{i-1}, s_i)$. To define the contextual edit distance between arbitrary strings s_x and s_y one considers the minimum of $\text{CED}(\alpha)$ among all sequence of strings $\alpha = s_0, s_1, \ldots, s_k$ as above such that $s_0 = s_x$, $s_k = s_y$. That is, $\text{CED}(s_x, s_y) = \min\left\{ \text{CED}(\alpha) \;\middle|\; \alpha = (s_0, s_1, \ldots, s_k),\ s_0 = s_x,\ s_k = s_y,\ \text{ED}(s_i, s_{i+1}) = 1 \right\}$.

3.2 Comparison to other edit distance functions

Comparing NED and ED is easy. The NED distance (like CED and GED) measures the average number edits, not just the total count. To see why this is needed, consider two short words x_1, x_2 that differ in k letters and two long word y_1, y_2 that also differ in k letters. In the context of software verification, for example, the latter represent runs that are more similar to one another than the former. We thus, expect the distance between the ys to be less than the distances between the xs but this is not the case in ED, as can be observed by inspecting the following words.

$$\text{ED}(aabcde, abpcg) = 4 \qquad \text{NED}(aabcde, abpcg) = 4/7$$
$$\text{ED}(a^{96}b^4, a^{100}) = 4 \qquad \text{NED}(a^{96}b^4, a^{100}) = 4/100$$

We turn to a comparisons of NED with the other normalized edit distances, GED and CED. Usually, being normalized means that the values of the distance functions are bounded within a given range, but this is not always the case. The lower bound is clearly 0 for NED, GED, and CED, since they are metric. The upper value of NED and GED is 1 but the values for CED are not bounded:

▶ **Claim 11.** *The values of NED and GED cannot exceed 1 and may reach 1, the values of CED are unbounded.*

Proof. For NED the numerator is the weight of an edit path, which is always smaller than the denominator which is the length of the edit path, thus $\text{NED}(w_1, w_2) \leq 1$ for all $w_1, w_2 \in \Sigma^*$. Since $\text{NED}(\varepsilon, a) = 1$ the upper bound is 1.

For GED the numerator is twice the weight of the edit path, and the denominator is once the weight of the edit path, plus the sum of length of the strings which is at least the size of the edit path, thus clearly at least the weight of the edit path. This shows GED cannot exceed 1. The fact that $\text{GED}(\varepsilon, a) = 1$ shows that 1 is the upper bound.

To see why CED is not bounded consider the sequence of words $\{a^i\}_{i \in \mathbb{N}}$. That is, the sequence $\varepsilon, a, aa, aaa, \ldots$. We have that $\text{CED}(\varepsilon, a^i) = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{i}$. Thus $\text{CED}(\varepsilon, a^i)$ is the sum of the Harmonic sequence up to the ith element, and since the Harmonic sequence diverges, CED is unbounded. ◀

Towards the second property of metrics that we consider, recall that the first requirements of a metric, *identity of indiscernibles*, is that $d(s_1, s_2) = 0$ if and only if $s_1 = s_2$. That is, the distance between two strings (in our case) is zero if and only if it is the exact same string. In the case of strings, when working with a normalized distance with an upper bound 1, we

expect the distance to be 1, the maximal possible, if the strings are completely different, namely they do not have any letter in common, that is, for all $\sigma \in \Sigma$ if σ appears in s_1 it does not appear in s_2 and vice versa. In software verification, for example, this means that the system produced a run that is completely unrelated to the specification, thus we expect the distance to be 1, indicating it is as far away as possible from the specification.

Since CED is unbounded, we consider for the purpose of the next property, a slightly different version, that we call CED', defined as $\text{CED}'(s_1, s_2) = \min(1, \text{CED}(s_1, s_2))$.[2]

▶ **Property 12** (max variance of antitheticals). *Let $d\colon \Sigma^* \times \Sigma^* \to [0, 1]$ be an edit distance function. We say that d has the property of* max variance of antitheticals *if $d(s_1, s_2) = 1$ if and only if s_1 and s_2 have no letter in common.*

We show that NED has this property while GED and CED' do not.[3]

▶ **Claim 13.** *The property of max variance of antitheticals holds for NED, but does not hold for GED and CED'.*

Proof. Consider aa and bb. Since they have no common letter, we expect their distance to be 1. The fact that $\text{GED}(aa, bb) = 2/3$ shows that GED violates the property of max variance of antitheticals.[4] Consider a and $aaaa$. Since they do have a common letter, we expect their distance to be strictly less than 1. The fact that $\text{CED}'(a, aaaa) = 1$ shows that CED' violates the property of max variance of antitheticals.

To see that NED has this property, note that it results in a value of 1 iff the numerator equals the denominator, i.e., the weight of the edit path is the same as its length; which holds iff there are no edit letters with weight zero. Since the only zero weight edit letter is no-change, `n`, the value of NED is 1 if and only if the words have no common letter. ◀

For the third metric comparison property, consider two words u and v and suppose $d(u, v) = c$ for the concerned edit distance function d. When considering normalized edit distance, we expect that $d(u^i, v^i)$ will not exceed c since by repeating i times the edit operations for transforming u into v we should be able to transform u^i into v^i and the "average" number of edits will not change. It could be that when considering the longer words u^i and v^i there is a better sequence of edits, thus we do not expect equality. As before, our motivation for requiring this property comes from software verification. Specifically, when considering periodic runs, generated, e.g., by code with loops, one would expect that the distance between the periodic runs is not larger than the distance between the periods because an error that repeats regularly should only be counted once in a normalized measure that models average error rate.

▶ **Property 14** (Non escalation of repetitions). *Let d be an edit distance function. Let $u, v \in \Sigma^*$. If $d(u^k, v^k) \leq d(u, v)$ for any $k > 1$ we say that d* does not escalate repetitions.

▶ **Claim 15.** *The NED and GED distances satisfy the property of non escalation of repetitions. The CED and CED' distances do not.*

---

[2] This is inspired by [7] that explains this choice as follows: *"This measure is not normalized to a particular range. Indeed, for a string of infinite length and a string of 0 length, the contextual normalized edit distance would be infinity. But so long as the relative difference in string lengths is not too great, the distance will generally remain below 1.0"*.

[3] Note that extending this property to require that $d(s_1, s_2)$ equals the maximal value (be it 1 or more) only for antitheticals, so that it can be applied to the original CED, would not make CED satisfy it since $\text{CED}(\varepsilon, a) = 1 < \infty$.

[4] We note that, moreover, $\text{GED}(aab, b)$ is also $2/3$ though we expect $\text{GED}(aab, b) < \text{GED}(aa, bb)$ since the average number of edits is smaller in the first case.

Proof. Consider $u = aab$ and $v = aaab$. The following shows that CED and CED' escalate repetitions.

$$\text{CED}((aab)^1, (aaab)^1) = \frac{1}{4} = 0.25$$
$$\text{CED}((aab)^2, (aaab)^2) = \frac{1}{7} + \frac{1}{8} = \frac{15}{56} = 0.2678$$
$$\text{CED}((aab)^3, (aaab)^3) = \frac{1}{10} + \frac{1}{11} + \frac{1}{12} = \frac{181}{660} = 0.2742$$

To see that NED does not escalate repetitions, assume p_{uv} is an optimal edit path transforming u to v. Since $(p_{uv})^k$, the edit path obtained by repeating k times p_{uv}, is an edit path transforming u^k to v^k:

$$\text{NED}(u^k, v^k) \leq \frac{k \cdot wgt(p_{uv})}{k \cdot len(p_{uv})} = \frac{wgt(p_{uv})}{len(p_{uv})} = \text{NED}(u, v).$$

The same reasoning shows that GED does not escalate repetitions.

$$\text{GED}(u^k, v^k) \leq \frac{2k \cdot \text{ED}(u,v)}{k(|u|+|v|)+k \cdot \text{ED}(u,v)} = \frac{2 \cdot \text{ED}(u,v)}{|u|+|v|+\text{ED}(u,v)} = \text{GED}(u, v). \quad \blacktriangleleft$$

The last property we consider is referred to as *pure uniformity of operations*. While we assume the weights of delete, insert and substitution are uniform, the resulting edit distance function may not be purely uniform, in the following sense. Consider two strings s_1 and s_2 such that s_1 is shorter than s_2. Then to transform s_1 to s_2 we would need some insertion operations. Consider now a word s'_1 that is longer than s_1 but not longer than s_2 and is obtained by padding s_1 with some new letter σ_{new} in some arbitrary set of positions. Since insert and substitution weigh the same, we expect $d(s_1, s_2)$ to be equal to $d(s'_1, s_2)$.

To define this formally we use the following notations. Let $\Sigma' \subseteq \Sigma$ and $s \in \Sigma^*$ we use $\pi_{\Sigma'}(s)$ for the string obtained from s by leaving only letters in Σ'. For instance, if $\Sigma = \{a, b, c\}$ and $s = abcbacc$ then $\pi_{\{a,b\}} = abba$.

▶ **Property 16** (pure uniformity). *Let $\Sigma, \Sigma_1, \Sigma_2$ be disjoints alphabets, and let $s_1, s_2 \in \Sigma^*$. We call d* purely uniform *if $d(s_1, s_2) = \min\{d(s'_1, s'_2) \mid s'_i \in (\Sigma \uplus \Sigma_i)^*$ and $\pi_\Sigma(s'_i) = s_i$ for $i \in \{1, 2\}\}$.*

We can now show that NED satisfies this property while GED and CED do not.

▶ **Claim 17.** *The NED distance is purely uniform. The GED and CED distances are not.*

Proof. To see why GED and CED are not purely uniform consider the words $s_1 = a^{50}$, $s_2 = a^{100}$ and $s'_1 = a^{50}c^{50}$ and note that $\pi_{\{a,b\}}(s'_1) = s_1$. We have that $\text{GED}(a^{50}, a^{100}) = 2 \cdot 50/(150+50) = 1/2$ whereas $\text{GED}(a^{50}c^{50}, a^{100}) = 100/(200+100) = 1/3$. Considering CED, we have that $\text{CED}(a^{50}, a^{100}) = \sum_{i=51}^{100} \frac{1}{i} \approx 0.68817$ whereas $\text{CED}(a^{50}c^{50}, a^{100}) = \sum_{i=51}^{100} \frac{1}{100} = 0.5$. Since all values are below 1, the same is true for CED'.

To show that NED is purely uniform we first note that $s_1, s_2 \in \Sigma^*$ implies s_1, s_2 are in $(\Sigma \uplus \Sigma_1)^*$ and $(\Sigma \uplus \Sigma_2)^*$, respectively, thus the $\geq$ direction of the equality in Property 16 clearly holds. For the $\leq$ direction, we turn to Claim 18 below, which essentially formalized the intuition provided regarding the *alignment view* of NED. Thus, given s'_1 and s'_2 establishing the min in the RHS of Property 16, and $p' \in \Gamma^*$ an edit path transforming s'_1 into s'_2, we can build an edit path $p \in \Gamma^*$ transforming $\pi_\Sigma(s'_1)$ into $\pi_\Sigma(s'_2)$ such that $cost(p) \leq cost(p')$. This shows that $\text{NED}(s_1, s_2) \leq \text{NED}(s'_1, s'_2)$ for every such s'_1, s'_2. Thus NED satisfies the pure uniformity property. $\blacktriangleleft$

▶ **Claim 18.** *Let $\Sigma, \Sigma_1, \Sigma_2$ be disjoints nonempty alphabets. Let $s'_1 \in \Sigma \uplus \Sigma_1$ and $s'_2 \in \Sigma \uplus \Sigma_2$ and p' an edit path transforming s'_1 to s'_2. There exists an edit path p transforming $\pi_\Sigma(s'_1)$ to $\pi_\Sigma(s'_2)$ such that $cost(p) \leq cost(p')$.*

4 A Proof of the Triangle Inequality

This section is the main contribution of the paper – showing that NED with uniform costs satisfies the triangle inequality.

Let $s_1, s_2, s_3 \in \Sigma^*$ and p_{12}, p_{23} be edit paths, such that $apply(p_{12}, s_1) = s_2$, $apply(p_{23}, s_2) = s_3$. We would like to define a method $cmps\colon \Gamma_\Sigma^* \times \Gamma_\Sigma^* \to \Gamma_\Sigma^*$ that given the two edit paths p_{12}, p_{23} returns an edit path p_{13} from s_1 to s_3. In addition, using the notations $d_* = wgt(p_*)$ and $l_* = len(p_*)$ for $* \in \{12, 23, 13\}$, we would like to show that both of the following hold:

$$d_{13} \leq d_{12} + d_{23} \qquad (1) \qquad\qquad l_{13} \geq \max\{l_{12}, l_{23}\} \qquad (2)$$

From these two equations we can deduce that the cost of the resulting path p_{13} is at most the sum of costs of the given paths p_{12} and p_{23} proving that NED satisfies the triangle inequality.

Introducing a new edit letter. To do this we need, for technical reasons, to introduce a new edit letter, which we denote $\mathtt{b}$ (for *blank*). This is actually an abbreviation of $\mathtt{vx}$, that is, it signifies that a new letter is added and immediately deleted. We enhance the weight and length definition from Γ to $\Gamma \cup \{\mathtt{b}\}$ as follows.

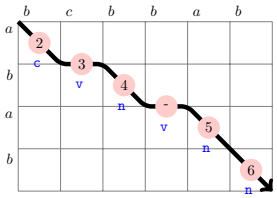
$$wgt(\gamma) = \begin{cases} 0 & \text{if } \gamma = \mathtt{n} \\ 1 & \text{if } \gamma \in \{\mathtt{c}, \mathtt{v}, \mathtt{x}\} \\ 2 & \text{if } \gamma = \mathtt{b} \end{cases} \qquad len(\gamma) = \begin{cases} 1 & \text{if } \gamma \in \{\mathtt{n}, \mathtt{c}, \mathtt{v}, \mathtt{x}\} \\ 2 & \text{if } \gamma = \mathtt{b} \end{cases}$$

As before we use the natural extensions of wgt and len from letters to strings and define $cost(p)$ to be $wgt(p)/\,len(p)$.

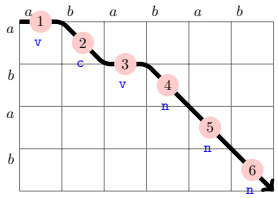
The compose method. We define a helper function $cmps_h$ that produces a string over $(\Gamma_\Sigma \cup \{\mathtt{b}\})^*$ (rather than over Γ_Σ^*). Given such a sequence we can convert it into a sequence over Γ_Σ by deleting all $\mathtt{b}$ symbols. The method $cmps_h\colon \Gamma_\Sigma^* \times \Gamma_\Sigma^* \to (\Gamma_\Sigma \cup \{\mathtt{b}\})^* \cup \{\perp\}$ is defined inductively, in Def. 19, by scanning the letters of the given edit paths p_{12}, p_{23}. We say that $cmps_h$ is well defined if it does not return $\perp$. We show that, when applied on edit paths p_{12} and p_{23} transforming some s_1 into s_2 and s_2 into s_3, respectively, $cmps_h$ is well defined.

▶ **Definition 19.** *Let p_{12}, p_{23} be edit paths over Γ_Σ. We define $cmps_h(p_{12}, p_{23})$ inductively as follows.*

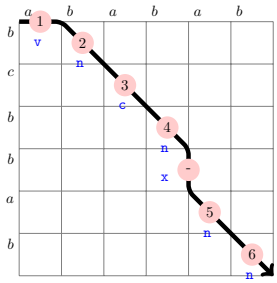
$$cmps_h(p_{12}, p_{23}) = \begin{cases} \varepsilon & \text{if } p_{12} = p_{23} = \varepsilon & (0) \\ \mathtt{x}_\sigma \cdot cmps_h(p_{12}[2..], p_{23}) & \text{if } p_{12}[1] = \mathtt{x}_\sigma & (1) \\ \mathtt{v}_\sigma \cdot cmps_h(p_{12}, p_{23}[2..]) & \text{if } p_{23}[1] = \mathtt{v}_\sigma & (2) \\ \mathtt{n}_\sigma \cdot cmps_h(p_{12}[2..], p_{23}[2..]) & \text{if } (p_{12}[1], p_{23}[1]) = (\mathtt{n}_\sigma, \mathtt{n}_\sigma) & (3) \\ \mathtt{c}_{(\sigma',\sigma)} \cdot cmps_h(p_{12}[2..], p_{23}[2..]) & \text{if } (p_{12}[1], p_{23}[1]) = (\mathtt{n}_{\sigma'}, \mathtt{c}_{(\sigma',\sigma)}) & (4) \\ \mathtt{x}_\sigma \cdot cmps_h(p_{12}[2..], p_{23}[2..]) & \text{if } (p_{12}[1], p_{23}[1]) = (\mathtt{n}_\sigma, \mathtt{x}_\sigma) & (5) \\ \mathtt{c}_{(\sigma_1,\sigma_3)} \cdot cmps_h(p_{12}[2..], p_{23}[2..]) & \text{if } (p_{12}[1], p_{23}[1]) = (\mathtt{c}_{(\sigma_1,\sigma_2)}, \mathtt{c}_{(\sigma_2,\sigma_3)}) & (6) \\ \mathtt{x}_{\sigma_1} \cdot cmps_h(p_{12}[2..], p_{23}[2..]) & \text{if } (p_{12}[1], p_{23}[1]) = (\mathtt{c}_{(\sigma_1,\sigma_2)}, \mathtt{x}_{\sigma_2}) & (7) \\ \mathtt{c}_{(\sigma',\sigma)} \cdot cmps_h(p_{12}[2..], p_{23}[2..]) & \text{if } (p_{12}[1], p_{23}[1]) = (\mathtt{c}_{(\sigma',\sigma)}, \mathtt{n}_\sigma) & (8) \\ \mathtt{v}_\sigma \cdot cmps_h(p_{12}[2..], p_{23}[2..]) & \text{if } (p_{12}[1], p_{23}[1]) = (\mathtt{v}_\sigma, \mathtt{n}_\sigma) & (9) \\ \mathtt{v}_{\sigma_2} \cdot cmps_h(p_{12}[2..], p_{23}[2..]) & \text{if } (p_{12}[1], p_{23}[1]) = (\mathtt{v}_{\sigma_1}, \mathtt{c}_{(\sigma_1,\sigma_2)}) & (10) \\ \mathtt{b} \cdot cmps_h(p_{12}[2..], p_{23}[2..]) & \text{if } (p_{12}[1], p_{23}[1]) = (\mathtt{v}_\sigma, \mathtt{x}_\sigma) & (11) \\ \perp & \text{otherwise} & (12) \end{cases}$$



(a) An optimal edit path p_{12} for w_1, w_2.



(b) The composed edit path p_{13} using Def. 19.



(c) An optimal edit path p_{23} for w_2, w_3.

$p_{12} = \texttt{cvnvnn}$

$p_{23} = \texttt{vncnxnn}$

$cmps_h(p_{12}, p_{23}) =$

$=$	$cmps_h(\texttt{cvnvnn}, \texttt{vncnxnn})$			
$=_1$	$\texttt{v} \cdot cmps_h(\texttt{cvnvnn}, \texttt{ncnxnn})$	case (2)	ε	$\texttt{v}_a$
$=_2$	$\texttt{vc} \cdot cmps_h(\texttt{vnvnn}, \texttt{cnxnn})$	case (8)	$\texttt{c}_{b,a}$	$\texttt{n}_b$
$=_3$	$\texttt{vcv} \cdot cmps_h(\texttt{nvnn}, \texttt{nxnn})$	case (10)	$\texttt{v}_c$	$\texttt{c}_{c,a}$
$=_4$	$\texttt{vcvn} \cdot cmps_h(\texttt{vnn}, \texttt{xnn})$	case (3)	$\texttt{n}_b$	$\texttt{n}_b$
$=_-$	$\texttt{vcvnb} \cdot cmps_h(\texttt{nn}, \texttt{nn})$	case (11)	$\texttt{v}_b$	$\texttt{x}_b$
$=_5$	$\texttt{vcvnbn} \cdot cmps_h(\texttt{n}, \texttt{n})$	case (3)	$\texttt{n}_a$	$\texttt{n}_a$
$=_6$	$\texttt{vcvnbnn}$	case (3)	$\texttt{n}_b$	$\texttt{n}_b$

$p_{13} = h(cmps_h(p_{12}, p_{23})) = h(\texttt{vcvnbnn}) = \texttt{vcvnnn}$

Figure 1 Let $w_1 = abab$, $w_2 = bcbbbab$, $w_3 = ababab$. Figure 1a shows an optimal edit path p_{12} between w_1 to w_2, Figure 1c shows an optimal edit path p_{23} between w_2 to w_3. Figure 1b shows the edit path p_{13} composed from p_{12} and p_{23} using Def. 19. The edit operations in Figure 1b are marked with numbers 1 to 6. A number n in between 1 and 6 in Figure 1a and Figure 1c signifies that the corresponding edge contributed to the construction of the edge marked n in Figure 1b (thus for the operations corresponding to cases (1) and (2) of Def. 19, there is one corresponding marking in Figure 1a and Figure 1c and for the others there are two). The labels $-$ in Figure 1a and Figure 1c correspond to case (11) dealing with adding a letter when going from s_1 to s_2 and deleting it when going from s_2 to s_3, which yields the edit symbol $\texttt{b}$. Note that p_{13} is not optimal; still its cost is better than the sum of the costs of p_{12} and p_{23}.

We further show that if the resulting string is p_{13} then applying the function *apply* to s_1 and the edit path obtained from p_{13} by deleting all $\texttt{b}$ results in the string s_3. Figure 1 shows an example of the application of $cmps_h$ on two given edit paths. In the sequel we will further show that the desired equations (Equation 1) and (Equation 2) hold.

Note that if we reach case (12) then we cannot claim that the result is an edit path. We thus first show that if $cmps_h$ is applied to two edit paths p_{12}, p_{23} such that $apply(p_{12}, s_1) = s_2$, and $apply(p_{23}, s_2) = s_3$, then the recursive application of $cmps_h(p_{12}, p_{23})$ will never reach the (12) case. That is, $cmps_h(p_{12}, p_{23})$ is well defined.

▶ **Lemma 20.** *Let $s_1, s_2, s_3 \in \Sigma^*$ and $p_{12}, p_{23} \in \Gamma_\Sigma^*$ be edit paths, such that $apply(p_{12}, s_1) = s_2$ and $apply(p_{23}, s_2) = s_3$. Then $p_{13} = cmps_h(p_{12}, p_{23})$ is well-defined.*

Proof. The proof is by structural induction on $cmps_h$. For the **base case**, we have that $p_{12} = p_{23} = \varepsilon$. Then $p_{13} = \varepsilon$. Thus $cmps_h$ reaches case (0) and is well defined.

For the **induction step** we have $p_{12} \neq \varepsilon$ or $p_{23} \neq \varepsilon$. If $p_{12} = \varepsilon$ then it follows from the definition of *apply* that $s_1 = s_2 = \varepsilon$. Given that $apply(p_{23}, \varepsilon)$ is defined we get that $p_{23}[1] = \texttt{v}_\sigma$. From the definition of *apply* we have $s_3 = \sigma \cdot apply(p_{23}[2..], s_2)$. Hence $s_3[2..] = apply(p_{23}[2..], s_2)$. Therefore, $cmps_h$ reaches case (2) and will never reach case (12) since from the induction hypothesis it follows that $cmps_h(p_{12}, p_{23}[2..])$ is well defined.

If $p_{23} = \varepsilon$ we get $s_2 = s_3 = \varepsilon$ and $p_{12}[1] = \texttt{x}_\sigma$. Hence $cmps_h$ reaches case (1) and similar reasoning shows that the induction hypothesis holds for the recursive application, and thus the result is well defined.

Otherwise the first character of p_{12} is not $\texttt{x}$ and the first character of p_{23} is not $\texttt{v}$. We consider the remaining cases, by examining first the first letter of p_{12}.

1. **Case** $p_{12}[1] = \mathtt{v}_{\sigma_1}$.
From the definition of *apply* we get that $s_2 = \sigma_1 \cdot s_2[2..]$ and $s_2[2..] = apply(p_{12}[2..], s_1)$.

   a. **Subcase** $p_{23}[1] = \mathtt{c}_{(\sigma_2,\sigma_3)}$.
   From the definition of *apply* it follows that $\sigma_1 = \sigma_2$, $s_3 = \sigma_3 \cdot s_3[2..]$ and $s_3[2..] = apply(p_{23}[2..], s_2[2..])$. Thus $cmps_h$ reaches case (10) and the induction hypothesis holds for the recursive application.
   b. **Subcase** $p_{23}[1] = \mathtt{n}_{\sigma_2}$.
   Similarly, from the definition of *apply* we get that $\sigma_1 = \sigma_2$, $s_3 = \sigma_2 \cdot s_3[2..]$ and furthermore $s_3[2..] = apply(p_{23}[2..], s_2[2..])$. Thus $cmps_h$ reaches case (9) and the induction hypothesis holds for the recursive application.
   c. **Subcase** $p_{23}[1] = \mathtt{x}_{\sigma_2}$.
   Similarly, from the definition of *apply* we get that $\sigma_1 = \sigma_2$ and $s_3 = apply(p_{23}[2..], s_2[2..])$. Thus $cmps_h$ reaches case (11) and the induction hypothesis holds for the recursive application.

2. **Case** $p_{12}[1] = \mathtt{c}_{(\sigma_1,\sigma_2)}$.
From the definition of *apply* we get that $s_1 = \sigma_1 \cdot s_1[2..]$, $s_2 = \sigma_2 \cdot s_2[2..]$ and furthermore $s_2[2..] = apply(p_{12}[2..], s_1[2..])$.

   a. **Subcase** $p_{23}[1] = \mathtt{c}_{(\sigma_3,\sigma_4)}$.
   From the definition of *apply* we get that $\sigma_2 = \sigma_3$, $s_3 = \sigma_4 \cdot s_3[2..]$ and $s_3[2..] = apply(p_{23}[2..], s_2[2..])$. Thus $cmps_h$ reaches case (6) and the induction hypothesis holds for the recursive application.
   b. **Subcase** $p_{23}[1] = \mathtt{n}_{\sigma_3}$.
   Similarly, from the definition of *apply* it follows that $\sigma_2 = \sigma_3$, $s_3 = \sigma_3 \cdot s_3[2..]$ and $s_3[2..] = apply(p_{23}[2..], s_2[2..])$. Thus $cmps_h$ reaches case (8) and the induction hypothesis holds for the recursive application.
   c. **Subcase** $p_{23}[1] = \mathtt{x}_{\sigma_3}$.
   Similarly, from the definition of *apply* we get that $\sigma_2 = \sigma_3$ and $s_3 = apply(p_{23}[2..], s_2[2..])$. Thus $cmps_h$ reaches case (7) and the induction hypothesis holds for the recursive application.

3. **Case** $p_{12}[1] = \mathtt{n}_{\sigma}$
From the definition of *apply* we get that $s_1 = \sigma \cdot s_1[2..]$, $s_2 = \sigma \cdot s_2[2..]$ and $s_2[2..] = apply(p_{12}[2..], s_1[2..])$.

   a. **Subcase** $p_{23}[1] = \mathtt{c}_{(\sigma_1,\sigma_2)}$.
   From the definition of *apply* it follows that $\sigma = \sigma_1$, $s_3 = \sigma_2 \cdot s_3[2..]$ and $s_3[2..] = apply(p_{23}[2..], s_2[2..])$. Thus $cmps_h$ reaches case (4) and the induction hypothesis holds for the recursive application.
   b. **Subcase** $p_{23}[1] = \mathtt{n}_{\sigma_2}$.
   Similarly, from the definition of *apply* it follows that $\sigma = \sigma_2$, $s_3 = \sigma_2 \cdot s_3[2..]$ and furthermore $s_3[2..] = apply(p_{23}[2..], s_2[2..])$. Thus $cmps_h$ reaches case (3) and the induction hypothesis holds for the recursive application.
   c. **Subcase** $p_{23}[1] = \mathtt{x}_{\sigma_2}$.
   Similarly, from the definition of *apply* we get that $\sigma = \sigma_2$ and $s_3 = apply(p_{23}[2..], s_2[2..])$. Thus $cmps_h$ reaches case (5) and the induction hypothesis holds for the recursive application. ◀

Recall that the $cmps_h$ returns a string over $\Gamma_\Sigma \cup \{\mathtt{b}\}$ while *apply* first argument is expected to be a string over Γ_Σ. We can convert the string returned by $cmps_h$ to a string over Γ_Σ by simply removing the $\mathtt{b}$ symbols. To make this precise we introduce the function $h\colon \Gamma_\Sigma \cup \{\mathtt{b}\} \to \Gamma_\Sigma$ defined as follows $h(\gamma) = \varepsilon$ if $\gamma = \mathtt{b}$ and $h(\gamma) = \gamma$ otherwise; and its natural extension $h\colon (\Gamma_\Sigma \cup \{\mathtt{b}\})^* \to \Gamma_\Sigma^*$ defined as $h(\gamma_1\gamma_2\cdots\gamma_n) = h(\gamma_1)h(\gamma_2)\cdots h(\gamma_n)$.

We are now ready to state that $cmps_h$ fulfills its task, namely if it returns p_{13} then $h(p_{13})$ is an edit path from s_1 to s_3 and its weight and length satisfy Equation 1 and Equation 2. Note that even if p_{12} and p_{23} are optimal, $h(p_{13})$ is not necessarily an optimal path from s_1 to s_3. Since the optimal path is no worse than $h(p_{13})$, it is enough for our purpose that $h(p_{13})$ is better than going through s_2.

▶ **Proposition 21.** *Let $s_1, s_2, s_3 \in \Sigma^*$ and p_{12}, p_{23} be edit paths, such that $apply(p_{12}, s_1) = s_2$, $apply(p_{23}, s_2) = s_3$. Let $p_{13} = cmps_h(p_{12}, p_{23})$. Let $d_* = wgt(p_*)$ and $l_* = len(p_*)$ for $* \in \{12, 23, 12\}$. Then the following holds*

1. $apply(h(p_{13}), s_1) = s_3$ **2.** $d_{13} \le d_{12} + d_{23}$ **3.** $l_{13} \ge \max\{l_{12}, l_{23}\}$

Proof. The proof is by structural induction on $cmps_h$. For the **base case**, we have that $p_{12} = p_{23} = \varepsilon$. Then $p_{13} = \varepsilon$, by definition of apply we get that $s_1 = s_2 = s_3 = \varepsilon$. Thus
1. $apply(h(p_{13}), s_1) = apply(\varepsilon, \varepsilon) = \varepsilon = s_1 = s_3$
2. and 3. we have that $d_{13} = 0 \le d_{12} + d_{23} = 0$ and $l_{13} = 0 \ge \max\{l_{12}, l_{23}\} = 0$

For the **induction steps**, we have $p_{12} \neq \varepsilon$ or $p_{23} \neq \varepsilon$. Recall that $p_{13} = cmps_h(p_{12}, p_{23})$. Thus, from Lem. 20 we can conclude p_{13} is a string over $\Gamma_\Sigma \cup \{\mathtt{b}\}$. Let $s'_* = s_*[2..]$, $p'_* = p_*[2..]$, $d'_* = wgt(p'_*)$, $l'_* = len(p'_*)$ for $* \in \{12, 23, 13\}$. The proof proceeds with the case analysis of $cmps_h$, going over cases (1)-(11) of Def. 19.

(1) Here $p_{12}[1] = \mathtt{x}_\sigma$.
Then from *apply* we have $s_1 = \sigma \cdot s'_1$, from definition of $cmps_h$ we have $p_{13} = \mathtt{x}_\sigma \cdot p'_{13}$
Since $s_2 = apply(p_{12}, s_1) = apply(\mathtt{x}_\sigma \cdot p'_{12}, \sigma \cdot s'_1) = apply(p'_{12}, s'_1)$ and $apply(p_{23}, s_2) = s_3$, by applying the induction hypotheses on s'_1, s_2, s_3 we get

1. $apply(h(p'_{13}), s'_1) = s_3$ 2. $d'_{13} \le d'_{12} + d_{23}$ 3. $l'_{13} \ge \max\{l'_{12}, l_{23}\}$

Therefore

1. $apply(h(p_{13}), s_1) = apply(\mathtt{x}_\sigma \cdot h(p'_{13}), \sigma \cdot s'_1) = apply(h(p'_{13}), s'_1) = s_3$
2. $d_{13} = 1 + d'_{13} \le 1 + d'_{12} + d_{23} = d_{12} + d_{23}$
3. $l_{13} = 1 + l'_{13} \ge 1 + \max\{l'_{12}, l_{23}\} \ge \max\{1 + l'_{12}, l_{23}\} = \max\{l_{12}, l_{23}\}$.

(2) Here $p_{23}[1] = \mathtt{v}_\sigma$.
Then from *apply* we have $s_3 = \sigma \cdot s'_3$, from definition of $cmps_h$ we have $p_{13} = \mathtt{v}_\sigma \cdot p'_{13}$.
Since $apply(p_{23}, s_2) = apply(\mathtt{v}_\sigma \cdot p'_{23}, s_2) = \sigma \cdot apply(p'_{23}, s_2) = s_3 = \sigma \cdot s'_3$ we get $apply(p'_{23}, s_2) = s'_3$ and $apply(p_{12}, s_1) = s_2$, by applying the induction hypotheses on s_1, s_2, s'_3 we get

1. $apply(h(p'_{13}), s_1) = s'_3$ 2. $d'_{13} \le d_{12} + d'_{23}$ 3. $l'_{13} \ge \max\{l_{12}, l'_{23}\}$.

Therefore

1. $apply(h(p_{13}), s_1) = apply(\mathtt{v}_\sigma \cdot h(p'_{13}), s_1) = \sigma \cdot apply(h(p'_{13}), s_1) = \sigma \cdot s'_3 = s_3$
2. $d_{13} = 1 + d'_{13} \le d_{12} + 1 + d'_{23} = d_{12} + d_{23}$
3. $l_{13} = 1 + l'_{13} \ge 1 + \max\{l_{12}, l'_{23}\} \ge \max\{l_{12}, 1 + l'_{23}\} = \max\{l_{12}, l_{23}\}$.

(3) Here $(p_{12}[1], p_{23}[1]) = (\mathtt{n}_\sigma, \mathtt{n}_\sigma)$.
From the definition of $cmps_h$ we have $p_{13} = \mathtt{n}_\sigma \cdot p'_{13}$ and from *apply* we have
$apply(p_{12}, s_1) = apply(\mathtt{n}_\sigma \cdot p'_{12}, \sigma \cdot s'_1) = \sigma \cdot apply(p'_{12}, s'_1) = \sigma \cdot s'_2 = s_2$ and
$apply(p_{23}, s_2) = apply(\mathtt{n}_\sigma \cdot p'_{23}, \sigma \cdot s'_2) = \sigma \cdot apply(p'_{23}, s'_2) = \sigma \cdot s'_3 = s_3$.

Since $apply(p'_{12}, s'_1) = s'_2$ and $apply(p'_{23}, s'_2) = s'_3$, by applying the induction hypotheses on s'_1, s'_2, s'_3 we get

1. $apply(h(p'_{13}), s'_1) = s'_3$ 2. $d'_{13} \leq d'_{12} + d'_{23}$ 3. $l'_{13} \geq \max\{l'_{12}, l'_{23}\}$.

Therefore

1. $apply(h(p_{13}), s_1) = apply(\mathtt{n}_\sigma \cdot h(p'_{13}), \sigma \cdot s'_1) = \sigma \cdot apply(h(p'_{13}), s'_1) = \sigma \cdot s'_3 = s_3$
2. $d_{13} = d'_{13} \leq d'_{12} + d'_{23} = d_{12} + d_{23}$
3. $l_{13} = 1 + l'_{13} \geq 1 + \max\{l'_{12}, l'_{23}\} = \max\{1 + l'_{12}, 1 + l'_{23}\} = \max\{l_{12}, l_{23}\}$.

(4) Here $(p_{12}[1], p_{23}[1]) = (\mathtt{n}_{\sigma'}, \mathtt{c}_{(\sigma',\sigma)})$.
By definition of compose we get $p_{13} = \mathtt{c}_{(\sigma',\sigma)} \cdot p'_{13}$. From *apply* we have

$apply(p_{12}, s_1) = apply(\mathtt{n}_{\sigma'} \cdot p'_{12}, \sigma' \cdot s'_1) = \sigma' \cdot apply(p'_{12}, s'_1) = \sigma' \cdot s'_2 = s_2$ and
$apply(p_{23}, s_2) = apply(\mathtt{c}_{(\sigma',\sigma)} \cdot p'_{23}, \sigma' \cdot s'_2) = \sigma \cdot apply(p'_{23}, s'_2) = \sigma \cdot s'_3 = s_3$.

Since $apply(p'_{12}, s'_1) = s'_2$ and $apply(p'_{23}, s'_2) = s'_3$, by applying the induction hypotheses on s'_1, s'_2, s'_3 we get

1. $apply(h(p'_{13}), s'_1) = s'_3$ 2. $d'_{13} \leq d'_{12} + d'_{23}$ 3. $l'_{13} \geq \max\{l'_{12}, l'_{23}\}$.

Therefore

1. $apply(h(p_{13}), s_1) = apply(\mathtt{c}_{(\sigma',\sigma)} \cdot h(p'_{13}), \sigma' \cdot s'_1) = \sigma \cdot apply(h(p'_{13}), s'_1) = \sigma \cdot s'_3 = s_3$
2. $d_{13} = 1 + d'_{13} \leq d'_{12} + 1 + d'_{23} = d_{12} + d_{23}$
3. $l_{13} = 1 + l'_{13} \geq 1 + \max\{l'_{12}, l'_{23}\} = \max\{1 + l'_{12}, 1 + l'_{23}\} = \max\{l_{12}, l_{23}\}$.

(5) Here $(p_{12}[1], p_{23}[1]) = (\mathtt{n}_\sigma, \mathtt{x}_\sigma)$.
By definition of compose we get that $p_{13} = \mathtt{x}_\sigma \cdot p'_{13}$. From *apply* we have

$apply(p_{12}, s_1) = apply(\mathtt{n}_\sigma \cdot p'_{12}, \sigma \cdot s'_1) = \sigma \cdot apply(p'_{12}, s'_1) = \sigma \cdot s'_2 = s_2$ and
$apply(p_{23}, s_2) = apply(\mathtt{x}_\sigma \cdot p'_{23}, \sigma \cdot s'_2) = apply(p'_{23}, s'_2) = s_3$.

Since $apply(p'_{12}, s'_1) = s'_2$ and $apply(p'_{23}, s'_2) = s_3$, by applying the induction hypotheses on s'_1, s'_2, s_3 we get

1. $apply(h(p'_{13}), s'_1) = s_3$ 2. $d'_{13} \leq d'_{12} + d'_{23}$ 3. $l'_{13} \geq \max\{l'_{12}, l'_{23}\}$.

Therefore

1. $apply(h(p_{13}), s_1) = apply(\mathtt{x}_\sigma \cdot h(p'_{13}), \sigma \cdot s'_1) = apply(h(p'_{13}), s'_1) = s_3$
2. $d_{13} = 1 + d'_{13} \leq d'_{12} + 1 + d'_{23} = d_{12} + d_{23}$
3. $l_{13} = 1 + l'_{13} \geq 1 + \max\{l'_{12}, l'_{23}\} = \max\{1 + l'_{12}, 1 + l'_{23}\} = \max\{l_{12}, l_{23}\}$.

(6) Here $(p_{12}[1], p_{23}[1]) = (\mathtt{c}_{(\sigma_1,\sigma_2)}, \mathtt{c}_{(\sigma_2,\sigma_3)})$.
By definition of compose we get that $p_{13} = \mathtt{c}_{(\sigma_1,\sigma_3)} \cdot p'_{13}$. From *apply* we have

$apply(p_{12}, s_1) = apply(\mathtt{c}_{(\sigma_1,\sigma_2)} \cdot p'_{12}, \sigma_1 \cdot s'_1) = \sigma_2 \cdot apply(p'_{12}, s'_1) = \sigma_2 \cdot s'_2 = s_2$ and
$apply(p_{23}, s_2) = apply(\mathtt{c}_{(\sigma_2,\sigma_3)} \cdot p'_{23}, \sigma_2 \cdot s'_2) = \sigma_3 \cdot apply(p'_{23}, s'_2) = \sigma_3 \cdot s'_3 = s_3$.

Since $apply(p'_{12}, s'_1) = s'_2$ and $apply(p'_{23}, s'_2) = s'_3$, by applying the induction hypotheses on s'_1, s'_2, s'_3 we get

1. $apply(h(p'_{13}), s'_1) = s'_3$ 2. $d'_{13} \leq d'_{12} + d'_{23}$ 3. $l'_{13} \geq \max\{l'_{12}, l'_{23}\}$.

Therefore

1. $apply(h(p_{13}), s_1) = apply(\mathtt{c}_{(\sigma_1,\sigma_3)} \cdot h(p'_{13}), \sigma_1 \cdot s'_1) = \sigma_3 \cdot apply(h(p'_{13}), s'_1) = \sigma_3 s'_3 = s_3$
2. $d_{13} = 1 + d'_{13} \leq 1 + d'_{12} + d'_{23} < 1 + d'_{12} + 1 + d'_{23} = d_{12} + d_{23}$
3. $l_{13} = 1 + l'_{13} \geq 1 + \max\{l'_{12}, l'_{23}\} = \max\{1 + l'_{12}, 1 + l'_{23}\} = \max\{l_{12}, l_{23}\}$.

(7) Here $(p_{12}[1], p_{23}[1]) = (\mathtt{c}_{(\sigma_1,\sigma_2)}, \mathtt{x}_{\sigma_2})$.
By definition of compose we get that $p_{13} = \mathtt{x}_{\sigma_1} \cdot p'_{13}$. From *apply* we have

$apply(p_{12}, s_1) = apply(\mathtt{c}_{(\sigma_1,\sigma_2)} \cdot p'_{12}, \sigma_1 \cdot s'_1) = \sigma_2 \cdot apply(p'_{12}, s'_1) = \sigma_2 \cdot s'_2 = s_2$ and
$apply(p_{23}, s_2) = apply(\mathtt{x}_{\sigma_2} \cdot p'_{23}, \sigma_2 \cdot s'_2) = apply(p'_{23}, s'_2) = s_3$.

Since $apply(p'_{12}, s'_1) = s'_2$ and $apply(p'_{23}, s'_2) = s_3$, by applying the induction hypotheses on s'_1, s'_2, s_3 we get

1. $apply(h(p'_{13}), s'_1) = s_3$ 2. $d'_{13} \leq d'_{12} + d'_{23}$ 3. $l'_{13} \geq \max\{l'_{12}, l'_{23}\}$.

Therefore

1. $apply(h(p_{13}), s_1) = apply(\mathtt{x}_{\sigma_1} \cdot h(p'_{13}), \sigma_1 \cdot s'_1) = apply(h(p'_{13}), s'_1) = s_3$
2. $d_{13} = 1 + d'_{13} \leq 1 + d'_{12} + d'_{23} < 1 + d'_{12} + 1 + d'_{23} = d_{12} + d_{23}$
3. $l_{13} = 1 + l'_{13} \geq 1 + \max\{l'_{12}, l'_{23}\} = \max\{1 + l'_{12}, 1 + l'_{23}\} = \max\{l_{12}, l_{23}\}$.

(8) Here $(p_{12}[1], p_{23}[1]) = (\mathtt{c}_{(\sigma',\sigma)}, \mathtt{n}_\sigma)$.
By definition of compose we get that $p_{13} = \mathtt{c}_{(\sigma',\sigma)} \cdot p'_{13}$. From *apply* we have

$apply(p_{12}, s_1) = apply(\mathtt{c}_{(\sigma',\sigma)} \cdot p'_{12}, \sigma' \cdot s'_1) = \sigma \cdot apply(p'_{12}, s'_1) = \sigma \cdot s'_2 = s_2$ and
$apply(p_{23}, s_2) = apply(\mathtt{n}_\sigma \cdot p'_{23}, \sigma \cdot s'_2) = \sigma \cdot apply(p'_{23}, s'_2) = \sigma \cdot s'_3 = s_3$.

Since $apply(p'_{12}, s'_1) = s'_2$ and $apply(p'_{23}, s'_2) = s'_3$, by applying the induction hypotheses on s'_1, s'_2, s'_3 we get

1. $apply(h(p'_{13}), s'_1) = s'_3$ 2. $d'_{13} \leq d'_{12} + d'_{23}$ 3. $l'_{13} \geq \max\{l'_{12}, l'_{23}\}$.

Therefore

1. $apply(h(p_{13}), s_1) = apply(\mathtt{c}_{(\sigma',\sigma)} \cdot h(p'_{13}), \sigma' \cdot s'_1) = \sigma \cdot apply(h(p'_{13}), s'_1) = \sigma \cdot s'_3 = s_3$
2. $d_{13} = 1 + d'_{13} \leq 1 + d'_{12} + d'_{23} = d_{12} + d_{23}$
3. $l_{13} = 1 + l'_{13} \geq 1 + \max\{l'_{12}, l'_{23}\} = \max\{1 + l'_{12}, 1 + l'_{23}\} = \max\{l_{12}, l_{23}\}$.

(9) Here $(p_{12}[1], p_{23}[1]) = (\mathtt{v}_\sigma, \mathtt{n}_\sigma)$.
By definition of compose we get that $p_{13} = \mathtt{v}_\sigma \cdot p'_{13}$. From *apply* we have

$apply(p_{12}, s_1) = apply(\mathtt{v}_\sigma \cdot p'_{12}, s_1) = \sigma \cdot apply(p'_{12}, s_1) = \sigma \cdot s'_2 = s_2$ and
$apply(p_{23}, s_2) = apply(\mathtt{n}_\sigma \cdot p'_{23}, \sigma \cdot s'_2) = \sigma \cdot apply(p'_{23}, s'_2) = \sigma \cdot s'_3 = s_3$.

Since $apply(p'_{12}, s_1) = s'_2$ and $apply(p'_{23}, s'_2) = s'_3$, by applying the induction hypotheses on s_1, s'_2, s'_3 we get

1. $apply(h(p'_{13}), s_1) = s'_3$ 2. $d'_{13} \leq d'_{12} + d'_{23}$ 3. $l'_{13} \geq \max\{l'_{12}, l'_{23}\}$.

1. $apply(h(p_{13}), s_1) = apply(\mathtt{v}_\sigma \cdot h(p'_{13}), s_1) = \sigma \cdot apply(h(p'_{13}), s_1) = \sigma \cdot s'_3 = s_3$
2. $d_{13} = 1 + d'_{13} \leq 1 + d'_{12} + d'_{23} = d_{12} + d_{23}$
3. $l_{13} = 1 + l'_{13} \geq 1 + \max\{l'_{12}, l'_{23}\} = \max\{1 + l'_{12}, 1 + l'_{23}\} = \max\{l_{12}, l_{23}\}$.

(10) Here $(p_{12}[1], p_{23}[1]) = (\mathtt{v}_{\sigma_1}, \mathtt{c}_{(\sigma_1,\sigma_2)})$.
By definition of compose we get that $p_{13} = \mathtt{v}_{\sigma_2} \cdot p'_{13}$. From *apply* we have

$apply(p_{12}, s_1) = apply(\mathtt{v}_{\sigma_1} \cdot p'_{12}, s_1) = \sigma_1 \cdot apply(p'_{12}, s_1) = \sigma_1 \cdot s'_2 = s_2$ and
$apply(p_{23}, s_2) = apply(\mathtt{c}_{(\sigma_1,\sigma_2)} \cdot p'_{23}, \sigma_1 \cdot s'_2) = \sigma_2 \cdot apply(p'_{23}, s'_2) = \sigma_2 \cdot s'_3 = s_3$.

Since $apply(p'_{12}, s_1) = s'_2$ and $apply(p'_{23}, s'_2) = s'_3$, by applying the induction hypotheses on s_1, s'_2, s'_3 we get

1. $apply(h(p'_{13}), s_1) = s'_3$ 2. $d'_{13} \leq d'_{12} + d'_{23}$ 3. $l'_{13} \geq \max\{l'_{12}, l'_{23}\}$.

Therefore

1. $apply(h(p_{13}), s_1) = apply(\mathtt{v}_{\sigma_2} \cdot h(p'_{13}), s_1) = \sigma_2 \cdot apply(h(p'_{13}), s_1) = \sigma_2 \cdot s'_3 = s_3$
2. $d_{13} = 1 + d'_{13} \leq 1 + d'_{12} + d'_{23} < 1 + d'_{12} + 1 + d'_{23} = d_{12} + d_{23}$
3. $l_{13} = 1 + l'_{13} \geq 1 + \max\{l'_{12}, l'_{23}\} = \max\{1 + l'_{12}, 1 + l'_{23}\} = \max\{l_{12}, l_{23}\}$.

(11) Here $(p_{12}[1], p_{23}[1]) = (\mathtt{v}_\sigma, \mathtt{x}_\sigma)$.
By definition of compose we get that $p_{13} = \mathtt{b} \cdot p'_{13}$. From *apply* we have

$apply(p_{12}, s_1) = apply(\mathtt{v}_\sigma \cdot p'_{12}, s_1) = \sigma \cdot apply(p'_{12}, s_1) = \sigma \cdot s'_2 = s_2$ and
$apply(p_{23}, s_2) = apply(\mathtt{x}_\sigma \cdot p'_{23}, \sigma \cdot s'_2) = apply(p'_{23}, s'_2) = s_3$.

Since $apply(p'_{12}, s_1) = s'_2$ and $apply(p'_{23}, s'_2) = s_3$, by applying the induction hypotheses on s_1, s'_2, s_3 we get

1. $apply(h(p'_{13}), s_1) = s_3$ 2. $d'_{13} \leq d'_{12} + d'_{23}$ 3. $l'_{13} \geq \max\{l'_{12}, l'_{23}\}$.

Therefore

1. $apply(h(p_{13}), s_1) = apply(h(p'_{13}), s_1) = s_3$
2. $d_{13} = 2 + d'_{13} \leq 1 + d'_{12} + 1 + d'_{23} = d_{12} + d_{23}$
3. $l_{13} = 2 + l'_{13} > 1 + \max\{l'_{12}, l'_{23}\} = \max\{1 + l'_{12}, 1 + l'_{23}\} = \max\{l_{12}, l_{23}\}$. ◀

The sequel makes use of the following lemmas regarding non-negative integers d and l.

▶ **Lemma 22.** *If $d \leq l$ then $\frac{d+1}{l+1} \geq \frac{d}{l}$*

Proof. $\frac{d+1}{l+1} = \frac{l(d+1)}{l(l+1)} \geq \frac{d(l+1)}{l(l+1)} = \frac{d}{l}$. ◀

▶ **Lemma 23.** *If $d_{13} \leq d_{12} + d_{23}$ and $l_{13} \geq \max\{l_{12}, l_{23}\}$ then $\frac{d_{12}}{l_{12}} + \frac{d_{23}}{l_{23}} \geq \frac{d_{13}}{l_{13}}$.*

Proof. $\frac{d_{13}}{l_{13}} \leq \frac{d_{12}+d_{23}}{l_{13}} = \frac{d_{12}}{l_{13}} + \frac{d_{23}}{l_{13}} \leq \frac{d_{12}}{l_{12}} + \frac{d_{23}}{l_{23}}$. ◀

Recall that *cost* is defined as *wgt* divided by *len*. Let p_{13} be the string obtained by compose in Prop. 21. Then by items 2 and 3 we know that

$$wgt(p_{13}) \leq wgt(p_{12}) + wgt(p_{23}) \qquad (3) \qquad len(p_{13}) \geq \max\{len(p_{12}), len(p_{23})\} \qquad (4)$$

We can thus conclude from Lem. 23 that the cost of the path obtained by $cmps_h$ is at most the sum of the costs of the edit paths from which it was obtained, as stated in the following corollary.

▶ **Corollary 24.** *Let $s_1, s_2, s_3 \in \Sigma^*$ and p_{12}, p_{23} be edit paths, such that $apply(p_{12}, s_1) = s_2$, $apply(p_{23}, s_2) = s_3$. Let $p_{13} = cmps_h(p_{12}, p_{23})$. Then $cost(p_{13}) \leq cost(p_{12}) + cost(p_{23})$.*

We are not done yet, since p_{13} contains `b` symbols, and thus it is not really an edit path. Let k be the number of `b`'s in p_{13}. Then $wgt(p_{13}) = 2k + wgt(h(p_{13}))$ and $len(p_{13}) = 2k + len(h(p_{13}))$, applying $2k$ times Lem. 22, we conclude that $\frac{wgt(p_{13})}{len(p_{13})} \geq \frac{wgt(h(p_{13}))}{len(h(p_{13}))}$.

▶ **Corollary 25.** $cost(p) \geq cost(h(p))$

▶ **Proposition 26.** *The normalized edit distance obeys the triangle inequality.*

Proof. Let $s_1, s_2, s_3 \in \Sigma^*$ and p_{12}, p_{23} be optimal edit paths. That is, $apply(p_{12}, s_1) = s_2$ and $apply(p_{23}, s_2) = s_3$ and $\text{NED}(s_1, s_2) = cost(p_{12})$ and $\text{NED}(s_2, s_3) = cost(p_{23})$. Let $p_{13} = cmps_h(p_{12}, p_{23})$. From Cor. 24 we get that $cost(p_{13}) \leq cost(p_{12}) + cost(p_{23})$. From Prop. 21 it holds that $h(p_{13})$ is a valid edit path over Γ_Σ. From Cor. 25 we get that $cost(h(p_{13})) \leq cost(p_{13})$. By definition of NED as it chooses the minimal cost of an edit path, $\text{NED}(s_1, s_3) \leq cost(h(p_{13}))$. To conclude, we get $\text{NED}(s_1, s_3) \leq \text{NED}(s_1, s_2) + \text{NED}(s_2, s_3)$. ◀

▶ **Theorem 27.** *The Normalized Levenshtein Distance NED (provided in Def. 4) with uniform costs (i.e., where the cost of all inserts, deletes and swaps are some constant c) is a metric on the space Σ^*.*

Proof. The first two conditions of being a metric follow from Prop. 7. The third condition, namely triangle inequality, follows from Prop. 26. ◀

5 Conclusions

We closed a gap regarding the normalized version of the editing distance proposed by Marzal and Vidal, denoted here as NED. Marzal and Vidal noted that NED is not a metric in general and left open the question of whether it is a metric in case all weights are equal. This open point, spawned two versions of a normalized editing distance that have been proven to be metrics – GED and CED. We proved that, with uniform weights, NED is also a metric. To pinpoint the benefits of NED over the other distances we have defined a number of properties that NED maintains and CED and/or GED do not. The motivation for formulating the properties as we did comes from formal verification, so is our interest in uniform weights.

References

1 Abdullah N Arslan and Omer Egecioglu. Efficient algorithms for normalized edit distance. *Journal of Discrete Algorithms*, 1(1):3–20, 2000.

2 Colin de la Higuera and Luisa Micó. A contextual normalised edit distance. In *Proceedings of the 24th International Conference on Data Engineering Workshops, ICDE 2008, April 7-12, 2008, Cancún, Mexico*, pages 354–361. IEEE Computer Society, 2008.

3 Emmanuel Filiot, Nicolas Mazzocchi, Jean-François Raskin, Sriram Sankaranarayanan, and Ashutosh Trivedi. Weighted transducers for robustness verification. In *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, pages 17:1–17:21, 2020.

4 Dana Fisman, Joshua Grogin, Oded Margalit, and Gera Weiss. The normalized edit distance with uniform operation costs is a metric. *CoRR*, abs/2201.06115, 2022. `arXiv:2201.06115`.

5 Vladimir Iosifovich Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, February 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

6 Yujian Li and Bi Liu. A normalized levenshtein distance metric. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1091–1095, 2007.

7 Christopher C. Little. `https://abydos.readthedocs.io/en/latest/abydos.distance.html#abydos.distance.HigueraMico`.

8 Andrés Marzal and Enrique Vidal. Computation of normalized edit distance and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):926–932, 1993.

9 Enrique Vidal, Andrés Marzal, and Pablo Aibar. Fast computation of normalized edit distances. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(9):899–902, 1995. `doi:10.1109/34.406656`.

A Appendix

We provide here two proofs that we could not fit in the body of the paper.

▶ **Proposition 7** (restated). *Let $s, s_1, s_2 \in \Sigma^*$. Then*

1. $NED(s, s) = 0$
2. *if* $s_1 \neq s_2$ *then* $NED(s_1, s_2) > 0$
3. $NED(s_1, s_2) = NED(s_2, s_1)$

Proof. First clearly, if $s \neq \varepsilon$ then $\mathtt{n}^{|s|}$ is an edit path from s to s, and thus $\text{NED}(s, s) = \frac{0}{|s|} = 0$. Second, if $s_1 \neq s_2$ then any edit path from s_1 to s_2 must contain at least one non-$\mathtt{n}$ character. Thus, its cost is $\frac{d}{l}$ for some $d > 0$, implying $\text{NED}(s_1, s_2) > 0$. Third, assume $p_{12} = \gamma_1\gamma_2 \dots \gamma_k$ is an edit path from s_1 to s_2. Define $\overline{p_{12}} = \overline{\gamma_1}\,\overline{\gamma_2} \dots \overline{\gamma_k}$ where

$$\overline{\gamma} = \begin{cases} \mathtt{n}_\sigma & \text{if } \gamma = \mathtt{n}_\sigma \\ \mathtt{c}_{(\sigma_2,\sigma_1)} & \text{if } \gamma = \mathtt{c}_{(\sigma_1,\sigma_2)} \\ \mathtt{x}_\sigma & \text{if } \gamma = \mathtt{v}_\sigma \\ \mathtt{v}_\sigma & \text{if } \gamma = \mathtt{x}_\sigma \end{cases}$$

Then $\overline{p_{12}}$ is an edit path from s_2 to s_1 and the cost they induce is the same. Hence, if p_{12} is a minimal edit path from s_1 to s_2 then $\overline{p_{12}}$ is a minimal edit path from s_2 to s_1 implying $\text{NED}(s_1, s_2) = \text{NED}(s_2, s_1)$. ◀

▶ **Claim 18** (restated). *Let $\Sigma, \Sigma_1, \Sigma_2$ be disjoints nonempty alphabets. Let $s'_1 \in \Sigma \uplus \Sigma_1$ and $s'_2 \in \Sigma \uplus \Sigma_2$ and p' an edit path transforming s'_1 to s'_2. There exists an edit path p transforming $\pi_\Sigma(s'_1)$ to $\pi_\Sigma(s'_2)$ such that $cost(p) \leq cost(p')$.*

Proof. Let $\gamma \in \Gamma$, $p' \in \Gamma^*_{\Sigma \cup \uplus \Sigma_1 \uplus \Sigma_2}$. We define $f : \Gamma_{\Sigma \uplus \Sigma_1 \uplus \Sigma_2} \to \Gamma_\Sigma$ as follows

$$f(\gamma) = \begin{cases} \mathtt{l}_\sigma & \text{if } \gamma = \mathtt{l}_\sigma \text{ for some } \mathtt{l} \in \{\mathtt{v}, \mathtt{x}, \mathtt{n}\} \text{ and } \sigma \in \Sigma \\ c_{\sigma,\sigma'} & \text{if } \gamma = c_{\sigma,\sigma'} \text{ and } \sigma, \sigma' \in \Sigma \\ \mathtt{v}_\sigma & \text{if } \gamma = c_{\sigma_1,\sigma} \text{ and } \sigma_1 \in \Sigma_1,\ \sigma \in \Sigma \\ \mathtt{x}_\sigma & \text{if } \gamma = c_{\sigma,\sigma_2} \text{ and } \sigma \in \Sigma,\ \sigma_2 \in \Sigma_2 \\ \varepsilon & \text{otherwise} \end{cases}$$

Let $p = f(p')$ where $f : \Gamma^*_{\Sigma \uplus \Sigma_1 \uplus \Sigma_2} \to \Gamma^*_\Sigma$ is the natural extension of f defined by $f(\gamma_1 \dots \gamma_m) = f(\gamma_1) \dots f(\gamma_n)$.

It is not hard to see that p is an edit path from $\pi_\Sigma(s'_1)$ to $\pi_\Sigma(s'_2)$. Since all removed edit operations have cost 1 we get from Lem. 22 that $cost(p) \leq cost(p')$ ◀