Ben-Gurion University of the Negev

The Faculty of Natural Sciences

The Department of Computer Science

A Normalized Edit Distance on Finite and Infinite Words

Thesis submitted in partial fulfillment of the requirements

for the Master of Sciences degree

Joshua Grogin

Under the supervision of **Prof. Dana Fisman and Prof. Gera Weiss**

February 2022

Ben-Gurion University of the Negev

The Faculty of Natural Sciences

The Department of Computer Science

A Normalized Edit Distance on Finite and Infinite Words

Thesis submitted in partial fulfillment of the requirements

for the Master of Sciences degree

Joshua Grogin

Under the supervision of **Prof. Dana Fisman and Prof. Gera Weiss**

Signature of student: _____ Date: _____

Signature of supervisor I: _____ Date: _____

Signature of supervisor II: _____ Date: _____

Signature of chairperson of the

committee for graduate studies: _____ Date: _____

February 2022

A Normalized Edit Distance on Finite and Infinite Words

Joshua Grogin

Master of Sciences Thesis

Ben-Gurion University of the Negev

2022

Abstract

Reactive systems [HP84] are computer systems that maintain a continuous interaction with the environment in which they execute. Examples of such systems are hardware devices embedded in cars or air-crafts, device drivers, operating systems and communication protocols. Such systems usually involve complex and challenging implementations which solve real-time constraints and concurrent programming. To ensure the safety and reliability of such critical systems, many resources are invested into verifying correctness of these critical systems. Formal verification is a common method of verifying whether a given system satisfies its requirements (a given property). Usually the system is converted into a simpler state machine (transition system) or (or symbolic representation of it) and the properties are represented in some temporal logic (a logic that reasons on truth values of propositions over a timeline). Verification methods such as model checking can mathematically prove whether the system meets its

requirements. Due to the nature of reactive systems which represent non-terminating systems, infinite words are usually used to describe runs of such systems. Classically, temporal logics output a Boolean value, that is, the system either satisfies a property or not. The classical approach is well studied but in reality, satisfaction is more of a spectrum. For instance, an aircraft's gyroscope must be accurate at all times but if it must fail, we would rather that happen at a high altitude rather near potential obstacles. Naturally, the concept of distance opens the possibility of asking further questions. For example, systems which can tolerate many errors are preferable over systems which collapse when small disturbances are introduced. Suppose for simplicity a run is represented by a binary string, then the tolerance question amounts to asking what is the minimal number of bit flips and/or bit losses that would render an accepting string rejecting. If a distance measure between strings is available the question can be articulated as "is there a run that is closer than a given threshold to not meeting the requirements?". In other words, we would like the distance to measure how much "disturbance" in a word we can afford without risking non-compliance. To answer such a question for runs of reactive systems, we need a distance measure between infinite words.

This thesis presents a metric for quantifying distances between infinite words that reflects deviations between words (runs) of implementation and words (runs) specified in the requirements. Specifically, we extend a normalized version [MV93] of the known edit distance (henceforth, NED) to a normalized edit distance on infinite words, which we term ω-NED. We provide algorithms for calculating distances between periodic words and between ω-regular languages. The latter can be used, for example, to measure the distance of an implementation from the set of undesired words, which gives a measure of the robustness of the implementation. It can also be used to measure the

worst-case deviation of an implementation from the requirements.

During the course of our research we were surprised to realize that the question whether NED is a metric in the uniform case is still open, and that various other normalized version of the edit distance were proposed to bypass this gap. Playing with the other normalized edit distance measures we concluded that NED is more natural in the case of infinite words and managed to prove it satisfies the triangle inequality (the requirement missing to prove it is a metric). Part I discusses the extension to infinite words and the algorithms for calculating distance, and Part II provides the proof of NED being a metric.

Acknowledgements

I would like to thank my advisors (and teachers), Dana Fisman and Gera Weies, first for introducing me to the wonderful world of formal-methods, Dana - giving me a strong theoretical background in Automata and Logic on Infinite Objects, Gere - introducing me to the practical uses and implications of Formal methods and Model checking, and for accepting me as a Master student and guiding me through this thesis. We spent endless time, correcting mistakes and reducing our questions to interesting and fun riddles. Most of all your patients along with passion really let me grow and learn while having fun doing so. Working with you really influenced my career both as a researcher and as an engineer.

I also want to thank Oded Margalit, for helping us crack the triangle inequality "riddle" and for your interesting input shared with Gera.

I have to thank Ben Gurion University for their wonderful preparatory engineering program, which gave me an opportunity and all the tools I need to express my love and passion for science.

Last but not least, I want to thank my immediate family for endless support, belief, love and faith through good and tough times. My mother for her magical way of raising us in the most creative environment, letting us think for our own, creating passionate critical thinking adults and of course an unforgettable childhood.

Contents

Chapter 1

Introduction

1.1 Formal verification

Our dependency on computer systems is growing in an unprecedented rate, we have come to a point where entire medical and banking operations rely on both hardware and software, to manage transfer and calculate our data, even our day to day ways of transportation rely on such systems. As we advance in technology, these systems are becoming increasingly complicated hereby virtually impossible to develop without a strong need to make sure all the parts are running properly. Failures in critical systems can lead to catastrophic events impacting the economy or even worse, endangering human lives. It is an obvious need and obligation to verify our systems and make sure they are running correctly. Testing is a significant part of any hardware/software development process, typically creating test cases and running/simulating the system over the different test cases and making sure we achieve the expected result. Since exhaustively testing all possible cases becomes infeasible for complicated systems, this

approach will report existing errors, but will certainly not find any errors which aren't in the set of test cases. Formal methods compliment the traditional testing by exploring the entire state-space of the system with respect to the specification, hereby either ensuring the system meets its specification or finding a counterexample. We can use a mathematical model to represent our system and its behaviour. A fascinating and yet confusing aspect of this approach, is that we are able to verify our system meets our specification without actually running our system, that is we prove our system is safe in contrast to testing our system on a subset of inputs.

The model checking problem asks, given a system and a temporal logic formula (the specification) [Pnu77, EF06], whether the system obeys the formula. A common method is to transform both the system and its specification into automata over infinite words, reducing the problem to checking whether the set of system traces is contained in the set of specification traces, or equivalently reduced to the emptiness problem, that is asking whether there exists a trace in the intersection between our system traces and the complementary of the property traces [Var95]. If our system doesn't meet the requirements of the property, we can retrieve the witness trace which doesn't meet the specification — a highly attractive quality of this approach since we get a counterexample which can be used as a test case for debugging our system [BK08, CGK$^+$18].

Recent research focuses on quantitative extensions of the model-checking problem where instead of Boolean acceptance (each word gets map to true or false) we get a quantitative value (each word gets mapped to a rational value). Naturally the classical Boolean decision problems (inclusion, universality, equivalence and emptiness) get a quantitative flavor [CDH10]. Noticeable work of determining distance between languages uses *simulation games*, where we can determine the distance of an imple-

mentation from the specification. We create a two player game where Player 1 (the implementation) chooses moves (transitions) and Player 2 (the specification) tries to match each move. The goal of Player 1 is to prove that simulation does not hold, by driving the game into a state from which Player 2 cannot match the chosen move. To make this a quantitative value we add 'cheat' transitions with a cost, and we look at the value of the infinite game. Different distance functions were defined for different purposes and in later papers these sorts of games were refined into *implication games* [CHR10, CHR12, CHOV17].

Filliot et al. [FMR$^+$20] introduce three measures of robustness of an implementation with respect to a specification, all based on weighted transducers [Ber79]. For instance, they ask what is the minimal threshold ν such that all words up to distance ν are still contained in the specification.

Tabuada and Neider suggested to extend *linear temporal logic* (LTL) [Pnu77] to a 5-valued logic, termed *robust LTL* (rLTL), that gives a mathematical meaning to violations of an LTL formula [TN16]. Having one truth value corresponding to true and four truth values corresponding to different shades of false. Since these values are well ordered, one can compare robustness between implementations even if they fail to meet their specifications. We hope to define a distance that enhances their intuition, and is also applicable for all ω-regular languages (due to the fact that LTL is non-counting [DG08], it does not cover all ω-regular properties, and nor does its extension, rLTL).

One well studied distance measure is *edit distance* [Lev66], in this thesis we consider extensions of the edit distance as a measure of distance between words and languages

1.2 Edit distance

The *edit distance* [Lev66], 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. For example the edit distance between the word a and b is 1 where the edit distance between $aaaaab$ and $aaab$ is 2. 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 [MV93] showed that this function, termed *post-normalized edit distance* in [MV93], 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 [dlHM08]. 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 short 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 thesis, 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.

Li and Liu [LL07] 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 Mi̇̀co [dlHM08] 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 section 6.3 we prove that NED, the original edit normalization approach proposed by Vidal and Marzal [MV93] 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.

In section 3.1 we extend NED to infinite words and rely on the open question which we resolve in the second part of this thesis. 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. [FMR+20] for computing the distance between regular sets of 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.

1.3 Contributions

We see our main contribution in defining a normalized distance function over infinite words. First we define a natural extension of the normalized edit distance from finite words to infinite words. We then prove that in the uniform case (where all the edit costs are equal) our extension is a metric on infinite words, or more precisely on equivalence classes of infinite words. Next we provide algorithms that demonstrate how the distance between two ultimately periodic words can be computed in polynomial time, and

how the distance between two regular ω-languages given by non-deterministic Büchi automata can be computed in polynomial time. When proving that our extended distance measure is a metric, we came across a question which has been open for more than 25 year — is the normalized edit distance proposed in [Marzal and Vidal 1993] a metric when the cost of all the edit operations are the same? Our main contribution has been submitted to LICS [FGW22]. The proof of the open gap in the literature was submitted to CPM [FGMW22].

Part I

Normalized edit distance on infinite words

Chapter 2

Preliminaries

In the next section we introduce the basic notations and some mathematical claims which we will utilize in the later proofs.

2.1 Notations and definitions

Sequences, sub-sequences, repetitions, projection We use $[i..j]$ to denote the set $\{i, i+1, \ldots, j\}$ for naturals i, j such that $i \leq j$. If $\rho = (r_0, r_1, \ldots)$ is a sequence, we use $\rho[i..j]$ for the sub sequence $(r_i, r_{i+1}, \ldots, r_j)$. Similarly, $\rho[i]$ is used to denote r_i, and $\rho[i..]$ is used to denote the suffix of ρ starting at i. If $\rho = (r_0, r_1, \ldots r_{l-1})$ is a sequence, we use ρ^k for the k-times repetition of ρ and ρ^ω for the infinite repetition of ρ. Then, if l is the length of ρ, we have that $\rho^\omega[i] = \rho[i \bmod l]$ for every $i \in \mathbb{N}$. Given a tuple $t = \langle a_1, a_2, \ldots, a_n \rangle$ we use $\pi_i(t)$ for a_i, namely the projection of t on the i-th coordinate. We extend this notions to sets, sequences and words in the usual manner, thus, for instance, given a sequence ρ=$(\langle \sigma_1, \sigma'_1 \rangle, \langle \sigma_2, \sigma'_2 \rangle \ldots)$ we use $\pi_1(\rho)$ for

the sequence $(\sigma_1, \sigma_2, \ldots)$.

Words, ω-words, ultimately periodic words, rotations An *alphabet* Σ is a finite non-empty set of symbols. A finite sequence over Σ is a word and an infinite sequence over Σ is an ω-word. We use $|w|$ to denote the length of w. Thus, $|w| = l$ if $w = \sigma_0\sigma_1 \ldots \sigma_{l-1}$ and $|w| = \omega$ if w is an ω-word. An ω word w is termed *ultimately periodic* if $w = uv^\omega$ for some $u \in \Sigma^*$ and $v \in \Sigma^+$. The set of finite words is denoted Σ^*, the set of infinite words is denoted Σ^ω, their union is denoted Σ^∞. The set of ultimately periodic words over Σ is denoted Σ^{UP}. Let $w \in \Sigma^*$, we use $rot(w)$ for the set of words uv such that $w = vu$, that is, all rotations of the word w.

Automata An *automaton* is a tuple $M = \langle \Sigma, Q, q_0, \delta \rangle$ consisting of a finite alphabet Σ of symbols, a finite set Q of states, an initial state q_0 and a transition function $\delta : Q \times \Sigma \to 2^Q$. A run of an automaton on a finite word $v = \sigma_1\sigma_2 \ldots \sigma_n$ is a sequence of states $q_0, q_1, \ldots, q_n$ such that $q_{i+1} \in \delta(q_i, \sigma_{i+1})$. A run on an infinite word is defined similarly and results in an infinite sequence of states. The transition function can be extended to a function from $Q \times \Sigma^*$ by defining $\delta(q, \lambda) = q$ and $\delta(q, \sigma v) = \delta(\delta(q, \sigma), v)$ for $q \in Q$, $a \in \Sigma$ and $v \in \Sigma^*$. We often use $M(v)$ as a shorthand for $\delta(q_0, v)$ and $|M|$ for the number of states in Q. A transition function is *deterministic* if $\delta(q, \sigma)$ is a singleton for every $q \in Q$ and $a \in \Sigma$, in which case we use $\delta(q, \sigma) = q'$ rather than $\delta(q, \sigma) = \{q'\}$.

Acceptors By augmenting an automaton with an acceptance condition α, obtaining a tuple $\langle \Sigma, Q, q_0, \delta, \alpha \rangle$, we get an *acceptor*, a machine that accepts some words and rejects others. An acceptor accepts a word, if one of the runs on that word is accepting. For finite words the acceptance condition is a set $F \subseteq Q$ and a run on v is accepting

if it ends in an accepting state, i.e. if $\delta(q_0, v) \in F$. For infinite words, there are many acceptance conditions in the literature, here we mention two: Büchi, the simplest one, and Muller, the most robust. A Büchi acceptance conditions is also a set $F \subseteq Q$. A run of a Büchi automaton is accepting if it visits F infinitely often. A Muller acceptance condition is a subset $\mathcal{F} = \{F_1, \ldots, F_k\}$ of subsets of Q. A run of a Muller automaton is accepting if the set S of states visited infinitely often along the run is exactly one of the subsets of $\mathcal{F}$. The set of words accepted by an acceptor $\mathcal{A}$ is denoted $[\![\mathcal{A}]\!]$. A language is said to be *regular* if it is accepted by a DFA. An ω-language is said to be *regular* if it is accepted by a deterministic Muller automaton.

Directions and Paths We consider a set $\mathbb{D} = \{(0,1), (1,0), (1,1)\}$ of three *directions*. The element $(0,1)$ denotes the *south* direction, and is abbreviated as d_{S}; the element $(1,0)$ the *east* direction, and is abbreviated as d_{E}; and the element $(1,1)$ the *south-east* direction, and is abbreviated as d_{SE}. A sequence $\rho \in \mathbb{D}^\infty$ is call a *path*.

A South-East Graph, Endpoint A *south-east graph* is composed of a set $V = [0..n_1] \times [0..n_2]$ of vertices, for some $n_1, n_2 \in \mathbb{N} \cup \{\omega\}$, and a set $E = (E_{\text{S}} \cup E_{\text{E}} \cup E_{\text{SE}}) \cap V^2$ of edges, where $E_{\text{S}} = \{(\langle i, j\rangle, \langle i, j+1\rangle)\}$, $E_{\text{E}} = \{(\langle i, j\rangle, \langle i+1, j\rangle)\}$, $E_{\text{SE}} = \{(\langle i, j\rangle, \langle i+1, j+1\rangle)\}$. We refer to such a south-east graph as *an* $(n_1 \times n_2)$*-south-east graph*. The vertices of the graph can be placed on a $[0..n_1] \times [0..n_2]$ grid. We visualize the top-left position as the $(0,0)$ point, and each edge in the graph corresponding to a move in one of the three directions. That is, a step in direction $d = (\delta_x, \delta_y) \in \mathbb{D}$ moves a token in point (i, j) to $(i + \delta_x, j + \delta_y)$. Therefore, a path $\rho = (d_0, \ldots, d_{l-1})$ starting in $(0,0)$ will end in $\sum_{i=0}^{l-1} d_i$ where sum is computed at each coordinate separately. Formally, we use $\mathit{endpoint}(\rho)$ to denote the endpoint of path ρ starting at $(0,0)$.

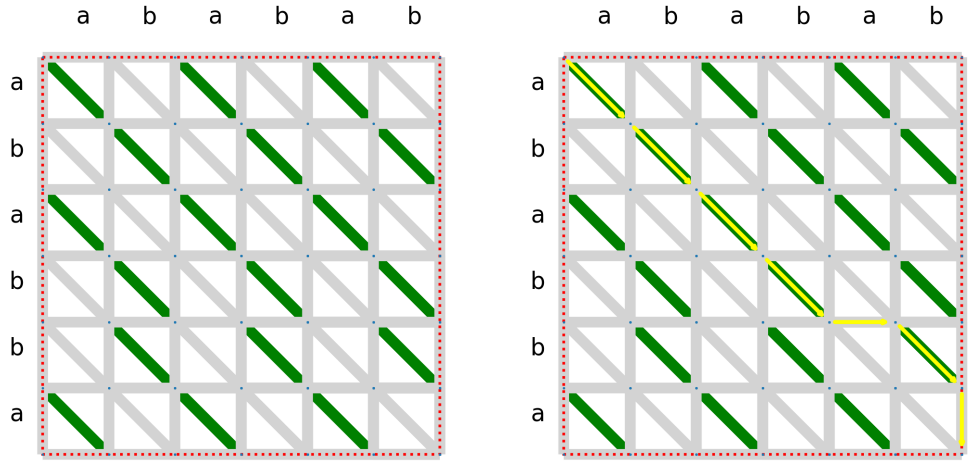


Figure 2.1: **Left**: the words graph $\mathcal{G} = \mathcal{G}(abababa, ababba)$, gray edges weigh 1 and green edges weigh 0. **Right**: a path ρ in $\mathcal{G}$ with $endpoint(\rho) = (6, 6)$, $wgt(\rho) = 2$, $len(\rho) = 7$, and $cost(\rho) = 2/7$.

Words Graph Given two words $w_1, w_2 \in \Sigma^\infty$ their corresponding graph $\mathcal{G}(w_1, w_2)$ is the weighted graph (G, θ) where $G = (V, E)$ is the $(|w_1| \times |w_2|)$-south-east graph and $\theta \colon E \to \{0, 1\}$ is defined as follows

$$\theta(e) = \begin{cases} 0 & \text{if } e{=}(v, v') \text{ for } v{=}(i, j), w_1[i]{=}w_2[j], v'{-}v{=}d_{\text{SE}}, \\ 1 & \text{otherwise} \end{cases}$$

That is, the weight of all east edges and south edges is 1 and a south-east edge starting at (i, j) will weigh 0 if the letters $w_1[i]$ and $w_2[j]$ are the same, and it will weigh 1 otherwise. The word graph $\mathcal{G}(abababa, ababba)$ is given in Figure 2.1 (Left).

Edit path A sequence $\rho \in \mathbb{D}^*$ is termed an *edit path* for (w_1, w_2) if $endpoint(\rho) = (|w_1|, |w_2|)$. The *weight* of ρ, denoted $wgt(\rho)$, is the sum of weights of the corresponding edges in $\mathcal{G}(w_1, w_2)$. Formally, if $\rho = (d_0, d_1, \ldots, d_{l-1})$ then the traversed edges are $(e_0, e_1, \ldots, e_{l-1})$ where $e_i = (s_{i-1}, s_{i+1})$, $s_{-1} = (0, 0)$ and $s_i = endpoint(\rho[..i])$ for $0 \leq i < l$. Hence, we can define $wgt(\rho) = \sum_{i=0}^{l-1} \theta(e_i)$. We use the notation $len(\rho)$ to

denote the length $|\rho|$ of ρ. The *cost* of ρ, denoted $cost(\rho)$, is defined to be $\frac{wgt(\rho)}{len(\rho)}$. See Figure 2.1 (Right). Intuitively, an edit path for (w_1, w_2) prescribes how to transform w_1 into w_2. In particular, a south direction from (i, j) marks that letter $w_1[i]$ is deleted, an east direction from (i, j) marks that letter $w_2[j]$ is added, a south-east direction from (i, j) marks substitution of $w_1[i]$ by $w_2[j]$, thus it costs nothing if $w_1[i] = w_2[j]$.

Edit Distance and the Normalized Edit Distance Using the above notations we provide the formal definitions of the *edit distance* and the *normalized edit distance*. The (not normalized) *edit distance* of $u_1, u_2 \in \Sigma^*$, denoted $\text{ED}(u_1, u_2)$, is the minimum weight of an edit path for (u_1, u_2). That is,

$$\text{ED}(u_1, u_2) = \min\{wgt(\rho) \mid \rho \text{ is an edit path for } (u_1, u_2)\}.$$

The *normalized edit distance* of two finite words $u_1, u_2 \in \Sigma^*$, denoted $\text{NED}(u_1, u_2)$, is the minimum cost of an edit path for (u_1, u_2). That is

$$\text{NED}(w_1, w_2) = \min\{cost(\rho) \mid \rho \text{ is an edit path for } (u_1, u_2)\}.$$

Since we are interested in the NED metric, we say that an edit path ρ for (u_1, u_2) is *optimal* if $\text{NED}(u_1, u_2) = cost(\rho)$.

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*.

Theorem 2.1 *[FGMW22]* (Σ^*, NED) *is a metric space.*

Limit functions Given a sequence $\eta = e_1, e_2, e_3, \ldots$ with $e_i \in \mathbb{R}$, we use the well known notations of limits for $\limsup, \liminf$ as follows:

$$\begin{aligned} \mathit{Sup}(\eta) &= \sup_{n\in\mathbb{N}} e_n \\ \mathit{Inf}(\eta) &= \inf_{n\in\mathbb{N}} e_n \\ \mathit{LimSup}(\eta) &= \limsup_{n\to\infty} e_n = \lim_{n\to\infty} \sup_{m\geq n} e_m \\ \mathit{LimInf}(\eta) &= \liminf_{n\to\infty} e_n = \lim_{n\to\infty} \inf_{m\geq n} e_m \end{aligned}$$

Note that for bounded sequences *LimSup* and *LimInf* always exist. We can also consider the well known summation functions of infinite sequences for taking the mean summation $\mathit{LimInfAvg}, \mathit{LimSupAvg}$ or discounted sum *Disc*

$$\begin{aligned} \mathit{LimSupAvg}(\eta) &= \limsup_{n\to\infty} \tfrac{1}{n} \textstyle\sum_{i=1}^{n} e_i \\ \mathit{LimInfAvg}(\eta) &= \liminf_{n\to\infty} \tfrac{1}{n} \textstyle\sum_{i=1}^{n} e_i \\ \mathit{Disc}_\lambda(\eta) &= \textstyle\sum_{n=1}^{\infty} \lambda^n e_n \text{ Where } 0 < \lambda < 1 \end{aligned}$$

2.2 Facts

We next prove facts that will be useful in late proofs.

Fact 2.2.1 *For every* $A, A', B, B' > 0$ *we have* $\frac{A+A'}{B+B'} \geq \min\{\frac{A}{B}, \frac{A'}{B'}\}$.

Proof. Assume w.l.o.g $\frac{A}{B} \leq \frac{A'}{B'}$. This implies $A' \geq \frac{A \cdot B'}{B}$. We get that

$$\begin{aligned} \frac{A}{B} = \frac{A(B+B')}{B(B+B')} = & \frac{A}{(B+B')} + \frac{A \cdot B'}{B(B+B')} \\ \leq & \frac{A}{(B+B')} + \frac{A'}{(B+B')} = \frac{A+A'}{B+B'} \end{aligned}$$

Fact 2.2.2 *Let $a_1, \ldots, a_n > 0$, $b_1, \ldots, b_n > 0$. Then*

$$\frac{\sum\limits_{1 \leq i \leq n} a_i}{\sum\limits_{1 \leq i \leq n} b_i} \geq \min\limits_{1 \leq i \leq n} \left\{ \frac{a_i}{b_i} \right\}.$$

Proof. The proof is by induction on m. The claim clearly holds for $m = 1$. For $m > 1$, let

$$A = \sum_{1 \leq i \leq n-1} a_i, \quad B = \sum_{1 \leq i \leq n-1} b_i, \quad A' = a_n, \quad B' = b_n,$$

By Fact 2.2.1 we have that

$$\frac{\sum\limits_{1 \leq i \leq n} a_i}{\sum\limits_{1 \leq i \leq mn} b_i} = \frac{A + A'}{B + B'} \geq \min \left\{ \frac{A}{B}, \frac{A'}{B'} \right\}.$$

By the induction hypothesis $\frac{A}{B} \geq \min\limits_{1 \leq i \leq n-1} \left\{ \frac{a_i}{b_i} \right\}$.
Hence, overall we get that

$$\frac{\sum\limits_{1 \leq i \leq n} a_i}{\sum\limits_{1 \leq i \leq mn} b_i} \geq \min\limits_{1 \leq i \leq n} \left\{ \frac{a_i}{b_i} \right\}.$$

□

Fact 2.2.3 *Let $d_t \colon \mathbb{W}^2 \to \mathbb{R}_+$ and $d_p \colon \mathbb{W}^2 \to \mathbb{R}_+$ be two metrics on $\mathbb{W}$. Define*

$d_{tp}\colon \mathbb{W}^2 \to (\mathbb{R}_+)^2$ *as follows*

$$d_{tp}(w_1, w_2) \stackrel{\text{def}}{=\!=} (d_t(w_1, w_2), d_p(w_1, w_2))$$

Then d_{tp} *is a metric on* $\mathbb{W}$ *when using the lex order.*

Fact 2.2.4 *Let* $d_u\colon \mathbb{U}^2 \to \mathbb{R}_+$ *and* $d_v\colon \mathbb{V}^2 \to \mathbb{R}_+$ *be two metrics. Define* $d_{uv} : (\mathbb{U} \times \mathbb{V})^2 \to (\mathbb{R}_+)^2$ *as follows*

$$d_{uv}(\langle u_1, v_1\rangle, \langle u_2, v_2\rangle) \stackrel{\text{def}}{=\!=} (d_u(u_1, u_2), d_v(v_1, v_2))$$

Then d_{uv} *is a metric on* $\mathbb{U} \times \mathbb{V}$ *when using the lex order.*

The proof of both facts are very similar, so we provide only the second one.

Proof. Let $u_x, u_y, u_z \in \mathbb{U}, v_x, v_y, v_z \in \mathbb{V}$. We show that the three conditions of a metric are satisfied.

- **Identity:**

$$d_{uv}(\langle u_x, v_x\rangle, \langle u_y, v_y\rangle) = (0, 0) \iff$$

$$d_u(u_x, u_y) = 0, d_v(v_x, u_v) = 0 \iff$$

$$u_x = u_y, v_x = v_y \iff$$

$$\langle u_x, v_x\rangle = \langle u_y, v_y\rangle$$

- **Symmetry:**

$$d_{uv}(\langle u_x, v_x\rangle, \langle u_y, v_y\rangle) = (d_u(u_x, u_y), d_v(v_x, v_y)) =$$

$$(d_u(u_y, u_x), d_v(v_y, v_x)) = d_{uv}(\langle u_y, v_y\rangle, \langle u_x, v_x\rangle)$$

- **Triangle inequality:**

  We have that

$$d_u(u_x, u_y) \leq d_u(u_x, u_z) + d_u(u_z, u_y)$$

$$d_v(v_x, v_y) \leq d_v(v_x, v_z) + d_v(v_z, v_y)$$

  And therefore

$$d_{uv}(\langle u_x, v_x\rangle, \langle u_y, v_y\rangle) = (d_u(u_x, u_y), d_v(v_x, v_y)) \leq$$

$$(d_u(u_x, u_z), d_v(v_x, v_z)) + (d_u(u_z, u_y), d_v(v_z, v_y)) =$$

$$d_{uv}(\langle u_x, v_x\rangle, \langle u_z, v_z\rangle) + d_{uv}(\langle u_z, v_z\rangle, \langle u_y, v_y\rangle)$$

Fact 2.2.5 *Given two sequences of positive numbers* $\vec{a} = (a_1, \ldots, a_m)$ *and* $\vec{b} = (b_1, \ldots, b_k)$ *satisfying* $\sum_{i=1}^{m} a_i = \sum_{j=1}^{k} b_j$ *there exists a matrix* $P \in [0,1]^{m\times k}$ *such that* $\sum_{i=1}^{m} p_{ij} = 1$ *for every* $1 \leq j \leq k$ *and* $\vec{a} = P \cdot \vec{b}$.
In other words, for all $1 \leq i \leq m$ *we have* $a_i = \sum_{j=1}^{k} p_{ij} \cdot b_j$.

Proof. Let

$$x = \sum_{i=1}^{m} a_i = \sum_{j=1}^{k} b_j.$$

Next, for all $1 \leq i \leq m, 1 \leq j \leq k$ let:

$$p_{ij} = \frac{a_i}{x} \in [0,1].$$

We get for all $1 \leq i \leq m$

$$\sum_{j=1}^{k} p_{ij} \cdot b_j = \sum_{j=1}^{k} \frac{a_i}{x} \cdot b_j = \frac{a_i}{\sum_{j=1}^{k} b_j} \sum_{j=1}^{k} b_j = a_i$$

and

$$\sum_{i=1}^{m} p_{ij} = \sum_{i=1}^{m} \frac{a_i}{x} = \frac{x}{x} = 1$$

Chapter 3

A Normalized Edit Distance for Infinite Words

In this chapter we present a normalized edit distance for infinite words, in the most natural way, i.e. the limit of NED for prefixes of the same length. We first discuss possibilities for metrics on infinite words. The most famous distance function on infinite words is the one on which the Cantor topology is defined, according to which the distance between $w_1, w_2 \in \Sigma^\omega$ decreases exponentially with the length of the longest common prefix [HR86]. Formally, using CTD to denote this distance function, $\text{CTD}(w_1, w_2)$ is zero if $w_1 = w_2$ and otherwise it is $2^{-\min\{n \mid w_1[..n] \neq w_2[..n]\}}$ where $w[..n]$ denotes the prefix of w of length n. Clearly, the intuition behind CTD and the edit distance functions mentioned above (ED and NED) is very different. We have that $\text{CTD}(ab^\omega, a^\omega) = 1/2$ and $\text{CTD}(ba^\omega, a^\omega) = 1$ while more edit operations are needed to get from ab^ω to a^ω than from ba^ω (in fact, infinitely many edit operations are required in the first case and only one in the second case).

Other commonly used distance functions for infinite words are defined using some weight function, and some summation function defined on that (see e.g.[CDH10, CHR12]). This is more similar in spirit to our motivation. We assume, for simplicity, that the weight function assigns uniform weights as above. Formally, for $w_1, w_2 \in \Sigma^\omega$ we define their uniform weight difference sequence as $\eta(w_1, w_2) = e_1, e_2, \ldots$ where $e_i = 0$ if $w_1[i] = w_2[i]$ and $e_i = 1$ otherwise. Further, given an infinite sequence $\eta = e_1, e_2, e_3, \ldots$ with $e_i \in \mathbb{R}$ the summation functions are defined as follows:

$$
\begin{aligned}
\mathit{Sup}(\eta) &= \sup_{n\in\mathbb{N}} e_n \\
\mathit{Inf}(\eta) &= \inf_{n\in\mathbb{N}} e_n \\
\mathit{LimSup}(\eta) &= \limsup_{n\to\infty} e_n = \lim_{n\to\infty} \sup_{m\geq n} e_m \\
\mathit{LimInf}(\eta) &= \liminf_{n\to\infty} e_n = \lim_{n\to\infty} \inf_{m\geq n} e_m \\
\mathit{LimSupAvg}(\eta) &= \limsup_{n\to\infty} \tfrac{1}{n} \textstyle\sum_{i=1}^{n} e_i \\
\mathit{LimInfAvg}(\eta) &= \liminf_{n\to\infty} \tfrac{1}{n} \textstyle\sum_{i=1}^{n} e_i \\
\mathit{Disc}_\lambda(\eta) &= \textstyle\sum_{n=1}^{\infty} \lambda^n e_n
\end{aligned}
$$

For finite words $\mathit{Sum}, \mathit{Min}, \mathit{Max}, \mathit{Avg}$ and Disc_λ are also used, see e.g., [FMR$^+$20]. Different summation functions are adequate in different situations. For instance, *Sup* is used for peak consumption; *LimSupAvg*, *LimInfAvg* for average response time or rate of failures; and Disc_λ when late failures are less important than early ones.

It is quite clear that *Sup*, *Inf*, *LimSup*, *LimInf*, and Disc_λ cannot be used to model edit distance in the flavor of ED and NED, but let's take it gradually. Applying *Sup* to $\eta(w_1, w_2)$ would give 1 if $w_1 \neq w_2$ and 0 otherwise. Applying *Inf* to $\eta(w_1, w_2)$ would give 1 only if $w_1[i] \neq w_2[i]$ for every i, i.e., they completely disagree. Clearly, these two are too coarse to model edit distance. Applying *LimSup* to $\eta(w_1, w_2)$ would give 1 if there are infinitely many indices i in which $w_1[i] \neq w_2[i]$ and 0 otherwise, and *LimInf* would

give 1 if there is i such that $w_1[j] \neq w_2[j]$ for every $j > i$. These are still too coarse for our purpose. The summation Disc_λ disregards what happens ad infinitum, or more precisely gives later events an exponentially smaller weight making them negligible. Even when considering words that agree on the suffixes, e.g., $w_1 = a^\omega$, $w_2 = aba^\omega$, $w_3 = aaaba^\omega$ we get that $\mathit{Disc}_{\frac{1}{2}}(\eta(w_1, w_2)) = 1/4$ and $\mathit{Disc}_{\frac{1}{2}}(\eta(w_1, w_3)) = 1/16$ though both w_2 and w_3 require one edit operation from w_1.

The summation functions closest to what we seek are *LimSupAvg* and *LimInfAvg*. Indeed, if we consider the words $w_1 = a^\omega$ and $w_2 = (aaaab)^\omega$, applying the uniform weight function $\eta(w_1, w_2)$ we obtain the sequence $\eta = (00001)^\omega$ since every fifth letter is different. Thus, $\mathit{LimSupAvg}(\eta) = 1/5$ which is consistent with our intuition on the normalized number of edit operations required to apply to w_1 in order to obtain w_2. For the words $v_1 = c^{100}a^\omega$ and $v_2 = d^{35}(aaaab)^\omega$ we also get the desired $1/5$ by applying *LimSupAvg* on $\eta(v_1, v_2)$ which is $1^{100}(00001)^\omega$. Indeed, *LimSupAvg* (and *LimInfAvg*) is indifferent to any finite prefix, as we expect the case to be.

Unfortunately, *LimSupAvg* and *LimInfAvg* do not always correspond to our intuition of the normalized number of edits required to get from one word to the other. Consider, e.g., $x_1 = (abc)^\omega$ and $x_2 = (acb)^\omega$ we have that $\eta(x_1, x_2) = (011)^\omega$ so *LimInfAvg* is $2/3$. But if we consider edit operations, we can transform x_1x_1 to x_2x_2 by $abcab\_c\_ \mapsto a\_c\_bacb$ to get that $\textsc{ned}(x_1x_1, x_2x_2)$=4/8=1/2, so we expect something similar for the ω-repetition.

Another issue arises when considering, for example, $y_1 = (abcd)^\omega$ and $y_2 = (bcda)^\omega$. The obtained sequence $\eta(y_1, y_2)$ is $(1)^\omega$ thus *LimSupAvg* and *LimInfAvg* will result in 1, meaning the words are as farthest apart from each other, while the number of edits required to get from y_1 to y_2 is one, since we can simply drop the first letter of y_1 to

get y_2. Since this is one edit out of infinitely many letters we expect a normalized edit distance on infinite words to return 0 in this case.

We now turn to discussing the desired criteria from an edit distance on infinite words. We want it to be *normalized* in the sense that the distance between two words is in $[0, 1]$. We would like it to reflect the number of edits needed in average to get from one word to the other. In particular, it would be nice if we can find such a metric in which the distance between $(u_1)^\omega$ and $(u_2)^\omega$ would be close to $\text{NED}(u_1, u_2)$. We would also like it to return zero for two words that have a common infinite suffix.

Do we want to require that the distance between two words is zero if and only if they have a common suffix? While this makes sense when considering ultimately periodic words, when we consider arbitrary words, there are more cases where we would like the distance to be zero. Consider, e.g., $w_1 = a^\omega$ and $w_2 = ba^9ba^{99}ba^{999}b\cdots$. That is, w_2 has a in almost all positions but b's creep in intervals of powers of 10. We do expect the distance between w_1 and w_2 to be 0 since the normalized number of required edits is negligible, in the sense that the necessity for an edit operation diminishes as the word progresses.

It is worth discussing that the issue in obtaining a distance function that meets the above criteria using the common summation functions may be in the error model (weight function) η defined above, which is quite naive and does not allow, for instance, deletions and insertions. In general, one can work with other error models, and indeed the literature, in general, allows arbitrary error models, typically encoded using a transducer [CHR12, FMR$^+$20]. The problem is that once an arbitrary error model is used, the chances it would be a metric get slimmer (as an example NED is not a metric when one allows non-uniform costs [MV93]). Thus, the challenge can be seen as finding

an error model with which we can prove a given distance function to be a metric (and meet the above criteria).

While our choice of ignoring finite prefixes has been justified above, in some cases one would like to distinguish for instance $z_1 = a^\omega$ and $z_2 = bbba^\omega$ and $z_3 = b^{100}a^\omega$ and require that the distance between z_1 and z_2 be smaller than the distance between z_1 and z_3. In particular, if the words have a common suffix (as is the case for z_1, z_2, z_3), we would like an edit distance that reflects the normalized number of edit operations required on the prefix. This requires a formal definition of when the prefix ends.

3.1 Extension to infinite words

Intuitively, it makes sense to define the *normalized edit distance* for two infinite words as the limit of NED of their prefixes. Since the limit may not exists, we define two versions, one using lim inf and one using lim sup as follows.

Definition 3.1.1 ($\overline{\omega\text{-ned}}$,$\underline{\omega\text{-ned}}$) *Let $w_1, w_2 \in \Sigma^\omega$ be two infinite words. We define two notions of a normalized edit distance, as follows:*

$$\overline{\omega\text{-NED}}(w_1, w_2) \stackrel{\text{def}}{=\!=} \limsup_{i\to\infty} \text{NED}(w_1[..i], w_2[..i])$$
$$\underline{\omega\text{-NED}}(w_1, w_2) \stackrel{\text{def}}{=\!=} \liminf_{i\to\infty} \text{NED}(w_1[..i], w_2[..i])$$

Since for every pair of finite words u_1, u_2, $\text{NED}(u_1, u_2)$ is bounded (between 0 and 1) both the $\overline{\omega\text{-NED}}(w_1, w_2)$ and $\underline{\omega\text{-NED}}(w_1, w_2)$ are well defined for every pair w_1, w_2 of infinite words.

Example 3.1.2

$$\overline{\omega\text{-NED}}(a^\omega,\ (aaaab)^\omega) = 1/5$$
$$\underline{\omega\text{-NED}}(a^\omega,\ (aaaab)^\omega) = 1/5$$

$$\overline{\omega\text{-NED}}(a^\omega,\ a\cdot b^1\cdot a\cdot b^2\cdot a\cdot b^3\cdot a\cdot b^4\cdots) = 1$$
$$\underline{\omega\text{-NED}}(a^\omega,\ a\cdot b^1\cdot a\cdot b^2\cdot a\cdot b^3\cdot a\cdot b^4\cdots) = 1$$

$$\overline{\omega\text{-NED}}(a^\omega,\ a^1\cdot b^1\cdot a^2\cdot b^2\cdot a^4\cdot b^4\cdots) = 1/2$$
$$\underline{\omega\text{-NED}}(a^\omega,\ a^1\cdot b^1\cdot a^2\cdot b^2\cdot a^4\cdot b^4\cdots) = 1/3$$

3.1.1 ω-ned is a metric

Recall that for a function to be a distance function it needs to satisfy the three conditions: (i) *identity of indiscernibles*, (ii) *symmetry*, and (iii) *triangle inequality*. Both $\overline{\omega\text{-NED}}$ and $\underline{\omega\text{-NED}}$ clearly satisfy the condition of symmetry.

Usually, the most challenging condition is the triangle inequality. We show that $\overline{\omega\text{-NED}}(w_1, w_2)$ satisfies the triangle inequality using the following claim:

Claim 3.1.3 *If* $d\colon \Sigma^*\times\Sigma^*\to\mathbb{R}_+$ *satisfies the triangle inequality then* $d_\omega\colon \Sigma^\omega\times\Sigma^\omega\to\mathbb{R}_+$ *where* $d_\omega(w_1, w_2)$ *is defined by* $\limsup_{i\to\infty} d(w_1[..i], w_2[..i])$ *satisfies the triangle inequality.*

Proof. Let $w_1, w_2, w_3 \in \Sigma^\omega$. We have that

$$\begin{aligned} d_\omega(w_1, w_3) &= \limsup_{i\to\infty} d(w_1[..i], w_3[..i]) \\ &\le \limsup_{i\to\infty} (d(w_1[..i], w_2[..i]) + d(w_2[..i], w_3[..i])) \\ &\le \limsup_{i\to\infty} d(w_1[..i], w_2[..i]) + \limsup_{i\to\infty} d(w_2[..i], w_3[..i]) \\ &= d_\omega(w_1, w_2) + d_\omega(w_2, w_3) \end{aligned}$$

where the first inequality holds since d satisfies the triangle inequality and the second inequality is a property of sum of lim sup of non-negative sequences. □

Corollary 3.1.4 *The function* $\overline{\omega\text{-NED}}\colon \Sigma^\omega \times \Sigma^\omega \to [0,1]$ *satisfies the triangle inequality.*

Note that the second inequality in the proof of Claim 3.1.3 does not hold if we replace lim sup by lim inf. Therefore, the above reasoning does not apply to $\underline{\omega\text{-NED}}$.

Regarding the condition of identity of indiscernibles we note, that as per the discussion in the introduction we do want to obtain a distance of 0 for some words $w_1 \neq w_2$. In other words we want the metric to be defined on equivalence classes of words, so that the distance between two words is zero if and only if they are in the same equivalence class.

To define this equivalence relation, let us revisit the examples where a distance of zero is expected. The first examples considered words that have a common suffix. Indeed in such pairs of words the number of required operations is finite (can be used to eliminate both prefixes), and thus negligible compared to the length of infinite words. The last example was $w_1 = a^\omega$ and $w_2 = ba^9ba^{99}ba^{999}b\cdots$. In this example we view the number of edits as negligible since the necessity for edit operations diminishes as the word progresses. In this example the number of required edits decreases exponentially. Should this be a requirement? What if it decreases quadratically or logarithmically? Observe that we can create words where the number of edits from a^ω decreases as slowly as desired by considering $w = (ab)^{n_1}(aab)^{n_2}(aaab)^{n_3}\cdots$. The larger $n_1, n_2, n_3, \ldots$ are the slower the number of edits decreases. Still, in all such words it diminishes over the infinite word and thus we expect the difference from a^ω to be zero.

We therefore define two infinite words w_1, w_2 to be *almost equal*, denoted $w_1 \equiv w_2$, if $\lim_{i\to\infty} \text{NED}(w_1[..i], w_2[..i]) = 0$. Note that words that have a common suffix are almost equal according to this definition, as are the other discussed examples. With this definition, $\overline{\omega\text{-NED}}$ satisfies the condition of identity of indiscernibles.

Claim 3.1.5 $\overline{\omega\text{-NED}}(w_1, w_2) = 0$ *iff* $w_1 \equiv w_2$.

Proof. If $w_1 \equiv w_2$ then $\lim_{i\to\infty} \text{NED}(w_1[..i], w_2[..i]) = 0$.
Thus $\limsup_{i\to\infty} \text{NED}(w_1[..i], w_2[..i]) = 0$.
Therefore, by definition $\overline{\omega\text{-NED}}(w_1, w_2) = 0$.

If $\overline{\omega\text{-NED}}(w_1, w_2) = 0$ then $\limsup_{i\to\infty} d(w_1[..i], w_2[..i]) = 0$.
Since $\text{NED}(v_1, v_2)$ is bounded between 0 and 1, for any $v_1, v_2 \in \Sigma^*$, it follows that $\liminf_{i\to\infty} d(w_1[..i], w_2[..i]) = 0$.
Hence $\lim_{i\to\infty} \text{NED}(w_1[..i], w_2[..i]) = 0$
implying $w_1 \equiv w_2$. □

Thus $\overline{\omega\text{-NED}}$ satisfies the three conditions of being a metric. on the space $\Sigma^\omega/\equiv$.

Theorem 3.6 $(\Sigma^\omega/\equiv, \overline{\omega\text{-NED}})$ *is a metric space.*

Observe that this entails that almost equal words can be used interchangeably when it comes to $\overline{\omega\text{-NED}}$. For $\underline{\omega\text{-NED}}$ we suffice with claiming it is ignorant of prefixes.

Claim 3.1.7 *For all* $w_1, w_2 \in \Sigma^\omega$, *if* $w_1' \equiv w_1$ *then* $\overline{\omega\text{-NED}}(w_1, w_2) = \overline{\omega\text{-NED}}(w_1', w_2)$. *If* $w_1' = w_1[i_0..]$ *for some* $i_0 \geq 0$ *then also* $\underline{\omega\text{-NED}}(w_1, w_2) = \underline{\omega\text{-NED}}(w_1', w_2)$.

Proof. For $\overline{\omega\text{-NED}}$, follows directly from the identity of indiscernibles and from the

triangle inequality. For $\underline{\omega\text{-NED}}$:

$$\begin{aligned}
\underline{\omega\text{-NED}}(w_1, w_2) &= \liminf_{i\to\infty} \text{NED}(w_1[..i], w_2[..i]) \\
&= \liminf_{i\to\infty} \text{NED}(w_1[..i_0 - 1]w_1[i_0..i], w_2'[..i]) \\
&= \liminf_{i\to\infty} \text{NED}(w_1'[..i + i_0], w_2'[..i]) \\
&= \liminf_{i\to\infty} \text{NED}(w_1'[..i], w_2'[..i]) \\
&= \underline{\omega\text{-NED}}(w_1', w_2')
\end{aligned}$$

The third equality is because an edit path from $w_1[..i]$ to $w_2[..i]$ can be constructed from an edit path from $w_1[i_0..i]$ to $w_2[..i]$ by adding i_0 delete operations at the beginning of the path. This means that the weight of the optimal paths cannot differ by more than i_0. Since the lengths of the edit paths grow indefinitely, this difference does not change the NED at the limit.

The fourth equality follows, similarly, because an edit path from $w_1'[..i]$ to $w_2'[..i]$ can be constructed from an edit path from $w_1'[..i + i_0]$ $w_2'[..i]$ by adding i_0 deletions at the end. □

The last two rows of Example 3.1.2 show that the lim inf and lim sup of the NED of the prefixes may, in general, converge to different numbers. In the next section we show that if w_1 and w_2 are ultimately periodic words then both notions converge to the same number, which can be calculated using NED of the best rotations of the periods (as formally stated in the sequel).

3.2 The Case of Ultimately Periodic Words

We turn to discuss *ultimately periodic words*. The interest in ultimately periodic words stems from the fact that (a) they provide a finite representation of an infinite word, so we can ask whether we can compute $\overline{\omega\text{-NED}}$ for such words (b) deterministic finite state machines generate ultimately periodic words and (c) two regular ω-languages are equivalent iff they agree on the set of ultimately periodic words.

We show that if w_1 and w_2 are ultimately periodic words, namely $w_1 = z_1 u_1^\omega$ and $w_2 = z_2 u_2^\omega$ for some $z_1, z_2 \in \Sigma^*$ and $u_1, u_2 \in \Sigma^+$ then $\overline{\omega\text{-NED}}(w_1, w_2) = \underline{\omega\text{-NED}}(w_1, w_2)$. Moreover, it equals the normalized-edit distance of the *best rotations* of u_1 with respect to u_2 (as defined next).

Definition 3.2.1 (best rotation) *Let $u_1, u_2 \in \Sigma^+$. Let $u_1' = u_1^\omega[..n]$ and $u_2' = u_2^\omega[..n]$ where n is the lowest common multiple of $|u_1|$ and $|u_2|$. We say that (v_1, v_2) is a* best rotation *for (u_1, u_2) if $v_1 \in rot(u_1')$, $v_2 \in rot(u_2')$ and for every $v_1' \in rot(u_1')$ and $v_2' \in rot(u_2')$ it holds that $\text{NED}(v_1, v_2) \leq \text{NED}(v_1', v_2')$. If (v_1, v_2) is a best rotation for some (u_1, u_2) we say that (v_1, v_2) are a* best rotation pair*. We refer to n as the* size *of the best rotation.*

Theorem 3.2 (ω-ned for ultimately periodic words) *Let $w_1 = z_1 u_1^\omega$ and $w_2 = z_2 u_2^\omega$. Let (v_1, v_2) be a best rotation for (u_1, u_2). Then $\overline{\omega\text{-NED}}(w_1, w_2) = \underline{\omega\text{-NED}}(w_1, w_2) = \text{NED}(v_1, v_2)$.*

The idea of the proof of Theorem 3.2 is as follows. By Claim 3.1.7 it suffices to consider the case where w_1 and w_2 are (completely) periodic. Say $w_1 = y_1^\omega$ and $w_2 = y_2^\omega$. Assume that (u_1, u_2) is a best rotation pair of (y_1, y_2) of size n. Consider first NED of the prefixes of u_1^ω and u_2^ω that are multiples of n. Let ρ be an optimal edit path for (u_1^i, u_2^i). Let

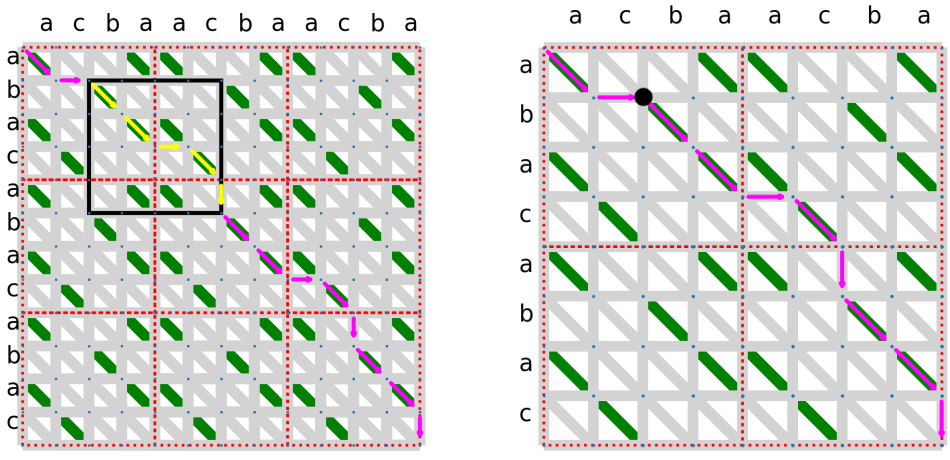


Figure 3.1: **Left:** $\mathcal{G}((acba)^3, (abac)^3)$ with an edit path ρ and points $s = (2, 1), t = (6, 5)$ such that $\rho[s..t]$ (in yellow) fits a 4-square. **Right:** the path $\rho' = \rho_s \cdot \rho_t$ obtained by removing from ρ the sub-path $\rho[s..t]$ (and replacing it by a black dot) as in the proof of Proposition 3.2.3.

ρ_* be an optimal path of (u_1, u_2). We show in Proposition 3.2.3 that the cost of ρ is not less than the cost of $(\rho_*)^i$, namely the cost of the edit path obtained from ρ_* by repeating it i times, which equals the cost of ρ_*. For prefixes that are not multiples of n, we show in Proposition 3.2.7 that their cost cannot be substantially better or worse than the cost of ρ_*.

Proposition 3.2.3 *Let $u_1, u_2 \in \Sigma^+$ be a best rotation pair of size n. Let ρ be an optimal edit path for (u_1^i, u_2^i) for some $i > 0$. Let ρ_* be an optimal path for (u_1, u_2). Then $cost(\rho) = cost(\rho_*)$.*

The proof of Proposition 3.2.3 builds on the following proposition, claiming that when looking at ρ on the words graph $\mathcal{G}(u_1^i, u_2^i)$, then some infix of ρ fits an $n \times n$ square, as formally stated in Proposition 3.2.4, and illustrated at the LHS of Figure 3.1.

Proposition 3.2.4 *Let $u_1, u_2 \in \Sigma^+$ be a best rotation pair of size n. Let ρ be an optimal edit path for (u_1^i, u_2^i) for some $i > 1$. Then there exists s and t, $0 \leq s < t < |\rho|$,*

such that $\mathit{endpoint}(\rho[s..t]) = (n, n)$.

The proof of Proposition 3.2.4 uses the intermediate value proposition over how much a $k \times k$ square drifts from the main diagonal.

Notice that Proposition 3.2.4 guarantees that along an optimal path ρ in the word graph $\mathcal{G}(u_1^i, u_2^i)$ for a best rotation pair (u_1, u_2) of size n there exists two points s and t such that $0 \leq s < t < |\rho|$ and $\mathit{endpoint}(\rho[s..t]) = (n, n)$. In order to prove it, we will prove a stronger claim (Lemma 3.2.6), that states that for any path (not necessarily optimal) that ends on some diagonal point (ℓ, ℓ) and any $0 \leq n \leq \ell$ i.e. not necessarily a n that divides ℓ (as is the case for n in Proposition 3.2.4 that divides $n \times i$), there exists such points s and t so that the twice replicated ρ^2 fits a square in between s and t, namely $\mathit{endpoint}(\rho^2[s..t]) = (n, n)$.

To prove, Lemma 3.2.6, the strengthening of Proposition 3.2.4 we introduce additional notations.

Path's drift Consider a path $\rho = d_0, d_1, \ldots, d_l$ starting at $(0, 0)$ and ending in some point on the diagonal, say (ℓ, ℓ).[1] Then $\mathit{endpoint}(\rho) = \sum_{i=0}^{l} d_i = (\ell, \ell)$. We would like to track the distance of the path from the diagonal, as well as the number of steps taken east or south, where d_{SE} is counted towards both. To this aim we associate with each direction d, a sequence $\tilde{d}$ of one or two elements as follows

$$\tilde{d} \stackrel{\text{def}}{=\!=} \begin{cases} (1) & \text{if } d = d_{\text{E}} \quad \text{i.e. } d = (1, 0) \\ (-1) & \text{if } d = d_{\text{S}} \quad \text{i.e. } d = (0, 1) \\ (0, 0) & \text{if } d = d_{\text{SE}} \quad \text{i.e. } d = (1, 1). \end{cases}$$

[1] Note that l and ℓ need not be the same.

Let $\hat{\rho}$ be the sequence obtained from concatenating $\tilde{d}_0, \tilde{d}_1, \ldots, \tilde{d}_l$. For instance, if $\rho = d_{\text{E}}, d_{\text{S}}, d_{\text{SE}}, d_{\text{SE}}$ then $\hat{\rho} = 1, -1, 0, 0, 0, 0$. Note that given ρ ends in (ℓ, ℓ) it must be that it makes ℓ moves east and ℓ moves south. Therefore $\hat{\rho}$ must be of length 2ℓ. Let $\hat{\rho} = \hat{d}_0, \hat{d}_1, \ldots, \hat{d}_{2\ell-1}$ for $\hat{d}_i \in \{-1, 0, 1\}$. Continuing the previous example, we get that ρ ends in $(3, 3)$, thus $\hat{\rho}$ is of length 6 and $\hat{d}_0 = 1$, $\hat{d}_1 = -1$, and $\hat{d}_i = 0$ for $i \in \{2, 3, 4, 5\}$. We use $\mathit{drift}(\rho)$ for the sum $\sum_{i=0}^{2\ell-1} \hat{d}_i$. Thus $\mathit{drift}(\rho)$ is zero if ρ makes the same number of south and east steps, it is positive if it makes more steps to the east than the south, and negative otherwise. Hence, if ρ ends in (ℓ, ℓ), implying it makes the same number of south and east steps, then $\mathit{drift}(\rho) = 0$.

An n-square on our path

We claim that if $\rho = d_0, d_1, \ldots, d_l$ is a path starting at $(0, 0)$ and ending at (ℓ, ℓ), then for every $0 \leq n \leq \ell$, looking at its twice replicated path ρ^2 there must exists two indices s and t on the path (where $0 \leq s < t < 2l$), such that $\mathit{endpoint}(\rho^2[s..t]) = (n, n)$. If we consider the south-east graph, looking at the path ρ^2, this means that we can find a point s on the path where we can place an $n \times n$ square and the path will intersect the left-top point and right-bottom point of the square (and the portion of the path within this square will be $\rho^2[s..t]$).

To prove this we consider sub-sequences of $\widehat{\rho^\omega}$ of length $2n$. Note that in general, not all sub-sequences of $\widehat{\rho}$ correspond to a sub-sequence of a path ρ, only those in which 0's come in consecutive pairs. If we consider an even length sub-sequence of $\widehat{\rho}$, then if it corresponds to a sub-sequence of ρ then it must be that the number of elements $\hat{d}_i$ that are either 1 or -1 is even. Hence the sum of its $\hat{d}_i$'s is even. Since we consider sub-sequences of $\widehat{\rho^2}$ of length $2n$, we define $\mathit{drift}_{2n}(m)$ of an index m on the path ρ

as $\mathit{drift}(\widehat{\rho^\omega}[m..m+2n-1])$, i.e., the drift of the $2n$-long sub-sequence of $\widehat{\rho^\omega}$ starting at index m. We say that index m is *legal* in $\hat{\rho}$ if $\mathit{drift}_{2n}(m)$ is even. Thus, an index m is legal if it corresponds to a sub-sequence of ρ. We will make use of the following lemma regarding illegal indices.

Lemma 3.2.5 *If index m is illegal then*

$$|\mathit{drift}_{2n}(m+1) - \mathit{drift}_{2n}(m)| \leq 1.$$

Proof. Let $\widehat{\rho^\omega} = \hat{d}_0, \hat{d}_1, \hat{d}_2, \ldots$. By the definition of drift_{2n} we have that

$$|\mathit{drift}_{2n}(m+1) - \mathit{drift}_{2n}(m)| = |\hat{d}_{m+2n-1} - \hat{d}_m|.$$

For the premise to be violated, i.e., for $|\hat{d}_{m+2n-1} - \hat{d}_m| > 1$ to hold, it must be that either $\hat{d}_{m+2n-1} = 1$ and $\hat{d}_m = -1$ or vice versa. But since m is illegal, either $\hat{d}_m$ is zero or $\hat{d}_{m+2n-1}$ is zero (as otherwise no subsequence of two consecutive zeros breaks). Contradiction. □

We can now prove the n-square proposition.

Lemma 3.2.6 (n-square) *Let $\rho = d_0, \ldots, d_l$ be a path starting at $(0,0)$ and ending at (ℓ, ℓ). Then, for every $0 \leq n \leq \ell$ there exist s and t such that $0 \leq s \leq l$, $s < t < 2l$ and*

$$\mathit{endpoint}(\rho^2[s..t]) = (n, n).$$

Proof. Recall that $\hat{\rho} = \hat{d}_0 \hat{d}_1 \ldots \hat{d}_{2\ell-1}$. Using the drift_{2n} notation, we have to show that there exists $0 \leq s \leq l$, such that $\mathit{drift}_{2n}(s) = 0$. Indeed, this implies that $\mathit{endpoint}(\rho^2[s..t]) = (n, n)$ where t is the index in ρ corresponding to the $s + 2n$ index

in $\widehat{\rho^2}$. Consider the sequence $\mathit{drift}_{2n}(0), \mathit{drift}_{2n}(1), \ldots, \mathit{drift}_{2n}(2\ell-1)$ and its sum. We have that

$$\sum_{s=0}^{2\ell-1} \mathit{drift}_{2n}(s) = 2n \cdot \mathit{drift}(\rho) = 0$$

since each $\hat{d}_i$ for $0 \leq i \leq 2\ell-1$ participates in exactly $2n$ sums on the left-hand side (because ρ is repeated twice). If $\mathit{drift}_{2n}(s) = 0$ for every $0 \leq s \leq l$ then the proposition clearly holds. Otherwise, there must exits $0 \leq m_1 < m_2 \leq 2\ell-1$ such that $\mathit{drift}_{2n}(m_1)$ is positive and $\mathit{drift}_{2n}(m_2)$ is negative or vice versa. If m_1 is illegal i.e. $\mathit{drift}_{2n}(m_1)$ is odd, then following Lemma 3.2.5 there must exist an index m_1' s.t. $m_1 < m_1' < m_2$ and $\mathit{drift}_{2n}(m_1')$ is even. Similarly if m_2 is illegal, then there must exist an index m_2' s.t. $m_1 < m_2' < m_2$ and $\mathit{drift}_{2n}(m_2')$ is even. Thus, we can assume without loss of generality that both m_1 and m_2 are legal.

Let $v_i = \mathit{drift}_{2n}(i)$ for $m_1 \leq i \leq m_2$. And consider the sequence obtained from $v_{m_1}, v_{m_1+1}, \ldots, v_{m_2}$ by removing all elements which are odd. Assume this sequence is $a_1, a_2, \ldots, a_t$ where $a_i = v_{m_1}$ and $a_t = v_{m_2}$. Then we have that this sequence satisfies that $a_{i+1} - a_i \in \{-2, 0, 2\}$. By the discrete version of the intermediate value [Joh98] we have that $\exists 0 \leq j \leq t$ such that $a_j = 0$. □

Now, using Proposition 3.2.4, we can show that the cost of an optimal edit path ρ for (u_1^i, u_2^i) is the same as the cost of ρ_*, an optimal for (u_1, u_2).

Proof (Proof of Proposition 3.2.3). Clearly, since $(\rho_*)^i$ is an edit path for (u_1^i, u_2^i) and ρ is defined to be optimal for (u_1^i, u_2^i) it must hold that $\mathit{cost}(\rho) \leq \mathit{cost}((\rho_*)^i) = \mathit{cost}(\rho_*)$.

The proof that $\mathit{cost}(\rho) \geq \mathit{cost}(\rho_*)$ is by induction on i. For $i = 1$, the path ρ clearly cannot cost less than the path ρ_* which is optimal for this dimension.

Consider $i > 1$. By Proposition 3.2.4 there exists $0 < s < n-1$ and $t > s$ such that

$\mathit{endpoint}(\rho[s..t]) = (n, n)$. Let $\rho_s = \rho[..s{-}1]$, $\rho_t = \rho[t{+}1..]$. Consider the path $\rho' = \rho_s \cdot \rho_t$ obtained by removing the sub-path of ρ from s to t (see Figure 3.1). It is an edit path for (u_1^{i-1}, u_2^{i-1}). Thus, by induction hypothesis we have that $\mathit{cost}(\rho') \geq \mathit{cost}(\rho_*)$. Since $\mathit{endpoint}(\rho[s..t]) = (n, n)$ we also have $\mathit{cost}(\rho[s..t]) \geq \mathit{cost}(\rho_*)$. Note that the cost of ρ can be computed using its sub-paths $\rho[s..t]$ and ρ' which combines ρ_s and ρ_t. Recall that the cost of a path is obtained by dividing the weight of the path by its length. Let $l' = \mathit{len}(\rho')$ and $d' = \mathit{wgt}(\rho')$. Similarly, let $l_{st} = \mathit{len}(\rho[s..t])$ and $d_{st} = \mathit{wgt}(\rho[s..t])$. Last, let $l_* = \mathit{len}(\rho_*)$ and $d_* = \mathit{wgt}(\rho_*)$. We get that

$$\begin{aligned} \mathit{cost}(\rho) = & \ \tfrac{\mathit{wgt}(\rho[s..t])+\mathit{wgt}(\rho')}{\mathit{len}(\rho[s..t])+\mathit{len}(\rho')} = \tfrac{d_{st}+d'}{l_{st}+l'} \\ \geq & \ \min\left\{\tfrac{d_{st}}{l_{st}}, \tfrac{d'}{l'}\right\} \geq \min\left\{\tfrac{d_*}{l_*}, \tfrac{d_*}{l_*}\right\} = \tfrac{d_*}{l_*} = \mathit{cost}(\rho_*) \end{aligned}$$

where the first inequality holds by Fact 2.2.2. □

We turn to bound, from above and below, the cost of prefixes that are not multiplication of n.

Proposition 3.2.7 *Let $u_1, u_2 \in \Sigma^+$ be a best rotation pair of size n. Let $m = i \cdot n + j$ for some $i > 0$ and $0 < j < n$ and let ρ be an optimal edit path for $(u_1^\omega[..m], u_2^\omega[..m])$. Let ρ_* be an optimal path for (u_1, u_2), and assume $d_* = \mathit{wgt}(\rho_*)$ and $l_* = \mathit{len}(\rho_*)$. Then $\frac{d_*}{l_*} - \frac{2(n-j)}{i \cdot n + j} \leq \mathit{cost}(\rho) \leq \frac{d_* + \frac{2j}{i}}{l_* + \frac{2j}{i}}$.*

Proof. For the left inequality, consider the path $\rho' = \rho \cdot (d_{\mathrm{S}})^{n-j} \cdot (d_{\mathrm{E}})^{n-j}$. That is, the path obtained from ρ by extending it with $(n - j)$ south and $(n - j)$ east steps. Note that $\mathit{endpoint}(\rho') = ((i{+}1)n, (i{+}1)n)$. It follows from Proposition 3.2.3 that the cost of ρ' is at least $\frac{d_*}{l_*}$. Thus we have

$$\mathit{cost}(\rho') = \frac{\mathit{wgt}(\rho) + 2(n-j)}{\mathit{len}(\rho) + 2(n-j)} \geq \frac{d_*}{l_*}$$

From here we get:

$$\begin{aligned} wgt(\rho) \cdot l_* &\geq d_* \cdot len(\rho) + d_* \cdot 2(n-j) - l_* \cdot 2(n-j) \\ &\geq d_* \cdot len(\rho) - l_* \cdot 2(n-j) \end{aligned}$$

dividing both sides by $l_* \cdot len(\rho)$ gives us

$$\frac{wgt(\rho)}{len(\rho)} \geq \frac{d_*}{l_*} - \frac{2(n-j)}{len(\rho)} \geq \frac{d_*}{l_*} - \frac{2(n-j)}{i \cdot n + j}$$

where the last inequality follows from the fact that $len(\rho) \geq i \cdot n + j$.

For the right inequality, consider the path $\rho' = (\rho_*)^i \cdot (d_{\mathrm{S}})^j \cdot (d_{\mathrm{E}})^j$. That is, the path obtained from ρ_*^i by extending it with j south and j east steps. Note that $endpoint(\rho') = (i \cdot n + j, i \cdot n + j)$. Since ρ is optimal we get:

$$cost(\rho) \leq cost(\rho') = \frac{wgt(\rho_*^i) + 2j}{len(\rho_*^i) + 2j} = \frac{i \cdot d_* + 2j}{i \cdot l_* + 2j} = \frac{d_* + \frac{2j}{i}}{l_* + \frac{2j}{i}}$$

We are now ready to prove Theorem 3.2 stating that for periodic words, the $\overline{\omega\text{-NED}}$ distance and $\underline{\omega\text{-NED}}$ are the same, and equivalent to the NED distance of their best rotation.

Proof (Proof of Theorem 3.2). Assume that $w_1 = z_1 u_1^\omega$ and $w_2 = z_2 u_2^\omega$. By Claim 3.1.7 $\overline{\omega\text{-NED}}(w_1, w_2) = \overline{\omega\text{-NED}}(u_1, u_2)$ and $\underline{\omega\text{-NED}}(w_1, w_2) = \underline{\omega\text{-NED}}(u_1, u_2)$. Let (v_1, v_2) be the best rotation of (u_1, u_2) of size n. Let ρ^* be an optimal path for v_1, v_2. Let d_* be its weight and l_* its length. Consider $\text{NED}(v_1^\omega[..i], v_2^\omega[..i])$. If i is a multiple of n then by Proposition 3.2.3 we get that $\text{NED}(v_1^\omega[..i], v_2^\omega[..i]) = d_*/l_*$. If i is not a multiple of n then by Proposition 3.2.7 we have that $\lim_{i\to\infty} \text{NED}(v_1^\omega[..i], v_2^\omega[..i]) = d^*/l^* = \text{NED}(v_1, v_2)$. □

Henceforth, to avoid cluttering, we use $\omega\text{-NED}(w_1, w_2)$ as a synonym for $\overline{\omega\text{-NED}}(w_1, w_2)$.

Chapter 4

Computing ω-ned for Languages of Infinite Words

We define the distance between two languages as the minimum distance between two words in the respective languages (as is common in metrics when extending the distance between pair of elements in the space to pair of sets in the space). Thus

$$\omega\text{-NED}(L_1, L_2) = \inf_{w_1 \in L_1, w_2 \in L_2} \omega\text{-NED}(w_1, w_2).$$

In this chapter we investigate the problem of computing ω-NED for regular languages of infinite words. Two questions that should be answered first, are

(a) how to compute ω-NED for two ultimately periodic words (which we discuss in section 4.1)

(b) how to compute NED for languages of finite words (which we discuss section 4.2).[1]

[1] Obviously, we define $\text{NED}(L_1, L_2) = \min_{w_1 \in L_1, w_2 \in L_2} \text{NED}(w_1, w_2)$.

After discussing these, in section 4.3 we tackle the problem of computing ω-NED for regular languages of infinite words.

4.1 Computing ω-ned for ultimately periodic words

We summarize first how computation of NED for finite words can be done.

Computing ned for words Computation of NED for two finite words can be done in PTIME using a dynamic programming algorithm as follows [MV93]. Let w_1, w_2 be words of lengths m, n and respectively. Any edit path of size k satisfies $\max\{m,n\} \leq k \leq m+n$. Thus, we can compute

$$D(i,j,k) = \min \left\{ wgt(\rho) \;\middle|\; \begin{array}{c} \rho \text{ is an edit path for } (w_1[..i-1], w_2[..j-1]) \\ \text{and } \mathit{len}(\rho) = k \end{array} \right\}$$

Therefore

$$\text{NED}(w_1, w_2) = \min_{max(m,n) \leq k \leq m+n} \frac{D(m,n,k)}{k}$$

and it can be computed in $O(m \cdot n \cdot \min\{m,n\})$, since for k we need $m+n-\max\{m,n\}$ entries.

Computing ω-ned for ultimately periodic words Let $w_1 = z_1 u_1^\omega$ and $w_2 = z_2 u_2^\omega$. Let $v_1 = u_1^\omega[..m]$ and $v_2 = u_2^\omega[..m]$ where m is the lowest common multiple of $|u_1|$ and $|u_2|$.

It follows from Theorem 3.2 that ω-NED(w_1, w_2) equals a best rotation (v_1', v_2') of (v_1, v_2). We note that to look for a best rotations it suffices to check only rotations of either v_1

or v_2.

Claim 4.1.1 *Let $v_1, v_2 \in \Sigma^*$ such that $|v_1| = |v_2|$. There exists a best rotation (v_1', v_2') of (v_1, v_2) where $v_2' = v_2$.*

Proof. Let (v_1', v_2') be a best rotation of (v_1, v_2) and let r be the rotation index in v_2 i.e.,

$$v_2' = v_2[r..] \cdot v_2[..r-1]$$

Consider the optimal edit path $\rho_* = (d_0, \ldots, d_k)$. Let i be the index such that $\mathit{endpoint}(\rho_*[..i]) = (x, n-r)$.
Let $\rho = \rho_*[i+1..k] \cdot \rho * [..i]$. Then ρ is an edit path for

$$(v_1'[x..n-1]v_1'[..x-1], v_2)$$

with $\mathit{cost}(\rho) = \mathit{cost}(\rho_*)$. □

Therefore ω-NED(w_1, w_2)= $\min_{v_1' \in rot(v_1)}$ NED(v_1', v_2) which can be computed in $O(m^4)$.

Corollary 4.1.2 *Let $w_1, w_2 \in \Sigma^{\text{UP}}$. Then ω-NED(w_1, w_2) can be computed in* PTIME.

4.2 Computing ned for regular languages

Filiot et al. [FMR$^+$20] have shown that the infimum of the mean of a weighted graph can be computed in PTIME.

Lemma 4.2.1 ([FMR$^+$20]) *Let $G = (V, E, W \colon E \to \mathbb{Q}_{\geq 0})$ be a weighted graph, and $V_I \subseteq V$, $V_F \subseteq F$ source and target vertices. The infimum of the mean weights of paths from V_I to V_F can be computed in* PTIME.

We can use this result to compute NED of regular languages, by building a weighted graph that corresponds to the product of two NFAs, where instead of allowing only transitions where both NFAs read the same letter, we also allow transitions where they read different letters, and transitions where one reads a letter and the other one does not (instead it reads ε). Transitions where both read a letter correspond to *replace*, where only the first reads a letter to *delete* and where only the second reads a letter to *insert*.

Definition 4.2.2 (Edit Distance Graph of two NFAs) *Let* $\mathcal{N}_i = (\Sigma, Q_i, s_i, \delta_i, F_i)$ *be a NFA for* $i \in \{1, 2\}$. *The* edit-distance graph of $\mathcal{N}_1$ and $\mathcal{N}_2$ *is a labeled weighted graph, denoted* $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ *which is defined as follows* $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2) = (V, L, E, \theta)$ *where* $V = Q_1 \times Q_2$ *is the set of vertices; the set of labels is* $\Sigma_\varepsilon \times \Sigma_\varepsilon$ *where* $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$*; the set of edges* $E \subseteq V \times L \times V$ *is given by*

$$E = \left\{ (\langle q_1, q_2 \rangle, \langle \sigma_1, \sigma_2 \rangle, \langle q_1', q_2' \rangle) \;\middle|\; \begin{array}{ll} \textit{either } \sigma_1 \neq \varepsilon \textit{ or } \sigma_2 \neq \varepsilon & \\ q_i' = \delta_i(q_i, \sigma_i) & \textit{if } \sigma_i \neq \varepsilon \\ q_i' = q_i & \textit{if } \sigma_i = \varepsilon \end{array} \right\}$$

and the weight function θ *associates a weight with edge* $(u, l, u') \in E$ *solely based on* l *as follows:*

$$\theta(u, l, u') = \theta_{\text{ED}}(l)$$

where

$$\theta_{\text{ED}}(\langle \sigma_1, \sigma_2 \rangle) = \begin{cases} 0 & \textit{if } \sigma_1 = \sigma_2 \\ 1 & \textit{otherwise} \end{cases}$$

Remark 4.2.3 *Note that we could have reduced the number of edges in* $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ *to include for any pair of nodes* (q_1, q_2) *and* (q_1', q_2')*, at most four edges: one corresponding*

to insert (if $\exists\sigma \in \Sigma, q_2 = \delta_2(q_2, \sigma), q_1' = q_1$*); one corresponding to delete (if* $\exists\sigma \in \Sigma, q_1 = \delta_1(q_1, \sigma), q_2' = q_2$*); one corresponding to swap (if* $\exists\sigma_1 \neq \sigma_2 \in \Sigma,\ q_i' = \delta_i(q_i, \sigma_i)$*); and one corresponding to swap of identical letters, i.e. no-op (if* $\exists\sigma \in \Sigma,\ q_i' = \delta_i(q_i, \sigma)$*). We choose to work with the version with more edges since it is more convenient for proofs.*

Clearly, any path in $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ corresponds to an edit path of the respective words and vice versa.

Claim 4.2.4 $\rho_{\text{ED}} = ((u_0, l_1, u_1),\ (u_1, l_2, u_2),\ \ldots, (u_{k-1}, l_k, u_k))$ *for* $l_i = (\sigma_i, \sigma_i')$ *is a path in* $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ *if and only if* $\rho = (d_1, d_2, \ldots, d_k)$ *is a path in the words graph* $\mathcal{G}(w_1, w_2)$ *where* $w_1 = \sigma_1\sigma_2\cdots\sigma_k$, $w_2 = \sigma_1'\sigma_2'\cdots\sigma_k'$ *and*

$$d_i = \begin{cases} d_{\text{S}} & \text{if } \sigma_i = \varepsilon \\ d_{\text{E}} & \text{if } \sigma_i' = \varepsilon \\ d_{\text{SE}} & \text{otherwise} \end{cases}$$

Note that in addition, all edges but those corresponding to swaps of identical letters cost 1. Thus, the sum of weights of a path ρ_{ED} in $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ corresponds to $wgt(|\rho|)$ and the length of ρ_{ED} to $len(|\rho|)$. Therefore, the infimum of the mean path in $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ corresponds exactly to the desired NED value. Hence, the NED distance between two regular languages languages given by NFAs $\mathcal{N}_1$ and $\mathcal{N}_2$ can be reduced to computing the infimum of the mean cycle in $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ from $V_I = \{(s_1, s_2)\}$ to $V_F = F_1 \times F_2$, and following Lemma 4.2.1 it can be computed in PTIME.

Theorem 4.5 *The* NED *distance between two regular languages given by NFAs* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *can be computed in* PTIME.

4.3 Computing ω-ned for Regular ω-Languages

The gist of the proof of Lemma 4.2.1 is important for the development of the algorithm for infinite words. The idea is that the infimum of the mean path in a weighted graph is obtained either on a simple path, or on a path with a simple cycle, by repeating the cycle over and over. Thus, one can use Karp's dynamic programing algorithm to compute the minimal mean value amongst simple paths and cycles, that works in PTIME [Kar78].

We would like to lift these ideas to compute ω-NED between two regular ω-languages, given by non-deterministic Büchi automata, henceforth NBA. To cope with the fact that we work with Büchi automata, we need to insist that the path has a cycle, and when projected on either components, visit an accepting state somewhere along the cycle (this will guarantee that the corresponding runs of both NBAs accept). Unfortunately, this alone does not suffice.

The problem is that if we find such a path with a cycle, which indeed corresponds to traversing infixes of the corresponding words, it might not correspond to infixes of the same length, as required by the definition of ω-NED. This is since the edit-distance graph has edges corresponding to deletes and insert, and such edges process letters only on one of the given NBA, not simultaneously on both. We should consider only lasso paths with the same number of delete and insert operations, as these are the paths that correspond to prefixes of the same length. To this aim, we add an additional annotation to edges, the so called *balance*.

Definition 4.3.1 (Edit Distance Graph of two NBAs) *Let* $\mathcal{B}_i = (\Sigma, Q_i, s_i, \delta_i, F_i)$ *be an NBA for* $i \in \{1, 2\}$. *The* edit-distance graph of $\mathcal{B}_1$ and $\mathcal{B}_2$ *is a labeled weighted graph, denoted* $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2) = (V, L', E, \theta)$, *which is defined similarly to the edit-distance*

graph for NFAs with one difference: the set of labels L' carries an additional annotation, the blanance. *Formally, $L' = L \times B$ where $L = \Sigma_\varepsilon \times \Sigma_\varepsilon$ is defined as for NFAs, $B = \{-1, 0, +1\}$, and label l is replaced by $(l, \mathit{bal}(l))$ where*

$$\mathit{bal}(l) = \begin{cases} +1 & \text{if } l \in \{\varepsilon\} \times \Sigma \\ -1 & \text{if } l \in \Sigma \times \{\varepsilon\} \\ 0 & \text{otherwise} \end{cases}$$

Given a path

$$\rho = v_0 \xrightarrow{l_1,b_1} v_1 \xrightarrow{l_2,b_2} \cdots \xrightarrow{l_t,b_t} v_t$$

in $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$ we use $\mathit{bal}(\rho)$ for $\sum_{i=1}^{t} b_i$.

With this definition, we are looking for the infimum mean lasso shaped path whose balance is zero and contains a cycle that visits both components F_1 and F_2 at least once. Note that in a lasso path, it is the weight of the cycle that matters, since by repeating the cycle as much as desired, the weight of the path until the cycle begins becomes negligible. Observe that the infimum mean balanced lasso path need not be simple, i.e., it can involve two or more cycles. E.g. if we have a cycle with balanced -2, another cycle with balance -3 and a cycle with balance 7 that share a vertex, the non-simple cycle repeating the -2 balanced cycle twice, and the other two cycles once is balanced. In this example since the cycles have a shared vertex, there is a cycle corresponding to the concatenation of one twice with the others. But, even if they do not share a vertex, that would be fine, because, again, by repeating them over and over again the paths that connect them become negligible. The next section shows that it suffices to consider lasso paths containing at most two simple cycles — either one (that is zero balanced) or two (one that is negatively balanced and one that is positively

balanced).

4.4 Enough to consider two cycles

Consider the graph $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$. Let $\mathcal{C} = \mathcal{C}_= \uplus \mathcal{C}_+ \uplus \mathcal{C}_-$ where

$$\mathcal{C}_= = \{\alpha \mid \alpha \text{ is a simple cycle and } \mathit{bal}(\alpha) = 0\}$$
$$\mathcal{C}_- = \{\alpha \mid \alpha \text{ is a simple cycle and } \mathit{bal}(\alpha) < 0\}$$
$$\mathcal{C}_+ = \{\alpha \mid \alpha \text{ is a simple cycle and } \mathit{bal}(\alpha) > 0\}$$

Here, $\mathcal{C}_=$ represents cycles that are zero balanced, $\mathcal{C}_+$ represents cycles that are positively balanced (have more inserts than deletes), and $\mathcal{C}_-$ those that are are negatively balanced. We define for a set of cycles $\alpha_1, \ldots, \alpha_n$, their value as follows.

Definition 4.4.1 (Value of a set of cycles) *Given a set of cycles $\{\alpha_1, \ldots, \alpha_n\}$, we use the following notations for $1 \leq i \leq n$:*

$$w_i = \theta(\alpha_i), \quad l_i = |\alpha_i|, \quad b_i = \mathit{bal}(\alpha_i).$$

We define

$$\mathit{val}(\alpha_1, \ldots, \alpha_n) = \min \left\{ \frac{\sum_{i=1}^n t_i \cdot w_i}{\sum_{i=1}^n t_i \cdot l_i} \;\middle|\; \begin{array}{l} \sum_{i=1}^n t_i \cdot b_i = 0 \\ t_i \geq 0 \text{ for } 1 \leq i \leq n \\ \exists i.\ t_i > 0 \end{array} \right\}$$

That is, $\mathit{val}(\alpha_1, \ldots, \alpha_n)$ computes the minimum among the NED value of the (imaginary) path obtained by concatenating all cycles, where cycle α_i is repeated t_i times,

such that the balance of the constructed path is zero. When we have one negatively balanced and one positively balanced cycles, their *val* can be easily computed, as follows.

Observation 4.4.2 *For* $\alpha_- \in \mathcal{C}_-, \alpha_+ \in \mathcal{C}_+$

$$val(\alpha_+, \alpha_-) = \frac{b_+ \cdot w_- - b_- \cdot w_+}{b_+ \cdot l_- - b_- \cdot l_+}$$

The following proposition states that from a set involving no zero balanced cycles, it is possible to choose just one positively balanced cycle and one negatively balanced cycle and the cost of the resulting path would not be worse than the one touring many cycles.

Proposition 4.4.3 *Let* $\alpha_1, \ldots, \alpha_m \in \mathcal{C}_-$, $\alpha_{m+1}, \ldots, \alpha_{m+k} \in \mathcal{C}_+$ *for some* $m, k \geq 1$. *There exists* $\alpha_+ \in \mathcal{C}_+$ *and* $\alpha_- \in \mathcal{C}_-$ *with*

$$val(\alpha_1, \ldots, \alpha_{m+k}) = val(\alpha_+, \alpha_-).$$

The proof makes use of Fact 2.2.5.

Proof (Proof of Proposition 4.4.3). Assume $t_1 \ldots t_{m+k} \in \mathbb{N}$ are an optimal solution for $val(\alpha_1, \ldots, \alpha_{m+k})$. Following Definition 4.4.1 the t_i's need to satisfy the following constraint:

$$0 \neq \sum_{i=1}^{m} t_i \cdot (-b_i) = \sum_{j=m+1}^{m+k} t_j \cdot b_j \tag{4.4.1}$$

By Fact 2.2.5 we can write Constraint 4.4.1 as the following m linear combinations,

one for each $1 \leq i \leq m$.

$$t_i \cdot (-b_i) = \sum_{j=m+1}^{m+k} \alpha_{ji} \cdot t_j \cdot b_j$$

such that $\alpha_{ji} \in [0,1]$ for all i, j; and $\sum_{i=1}^{m} \alpha_{ji} = 1$ for every j.

Following this observation and due to the optimal solution we get that $val(\alpha_1, \ldots, \alpha_{m+k}) =$

$$\begin{aligned}
&= \frac{\sum_{i=1}^{m+k} t_i \cdot w_i}{\sum_{i=1}^{m+k} t_i \cdot l_i} = \frac{\sum_{j=m+1}^{m+k} \sum_{i=1}^{m} t_j \cdot \alpha_{ji} \cdot (w_j - \frac{b_j}{b_i} \cdot w_i)}{\sum_{j=m+1}^{m+k} \sum_{i=1}^{m} t_j \cdot \alpha_{ji} \cdot (l_j - \frac{b_j}{b_i} \cdot l_i)} \\
&\geq \min_{m+1 \leq j \leq m+k, t_j \neq 0} \left\{ \frac{\sum_{i=1}^{m} t_j \cdot \alpha_{ji} \cdot (w_j - \frac{b_j}{b_i} \cdot w_i)}{\sum_{i=1}^{m} t_j \cdot \alpha_{ji} \cdot (l_j - \frac{b_j}{b_i} \cdot l_i)} \right\} \\
&\geq \min_{m+1 \leq j \leq m+k, t_j \neq 0} \left\{ \min_{1 \leq i \leq m, \alpha_{ji} \neq 0} \left\{ \frac{t_j \cdot \alpha_{ji} \cdot (w_j - \frac{b_j}{b_i} \cdot w_i)}{t_j \cdot \alpha_{ji} \cdot (l_j - \frac{b_j}{b_i} \cdot l_i)} \right\} \right\} \\
&= \min_{\substack{m+1 \leq j \leq m+k \\ 1 \leq i \leq m,\ \alpha_{ji}, t_j \neq 0}} \left\{ \frac{t_j \cdot \alpha_{ji} \cdot (w_j - \frac{b_j}{b_i} \cdot w_i)}{t_j \cdot \alpha_{ji} \cdot (l_j - \frac{b_j}{b_i} \cdot l_i)} \right\} \\
&= \min_{\substack{m+1 \leq j \leq m+k \\ 1 \leq i \leq m,\ \alpha_{ji}, t_j \neq 0}} \left\{ \frac{b_i \cdot w_j - b_j \cdot w_i}{b_i \cdot l_j - b_j \cdot l_i} \right\}
\end{aligned}$$

□

Henceforth, we use the following notations

$$\begin{aligned}
\mu_{=} &= \min_{\alpha \in \mathcal{C}_{=}} \{val(\alpha)\}, \\
\mu_{+-} &= \min_{\alpha_+ \in \mathcal{C}_+, \alpha_- \in \mathcal{C}_-} \{val(\alpha_+, \alpha_-)\}, \\
\mu_{*} &= \min_{\{\alpha_1, \ldots, \alpha_n\} \subseteq \mathcal{C}} \{val(\alpha_1, \ldots, \alpha_k)\}.
\end{aligned}$$

With these notations, we can conclude from Proposition 4.4.3 that μ_*, the best value

achieved for an arbitrary set of cycles, is no better than the best value achieved for one cycle or two.

Corollary 4.4.4 $\mu_* = \min\{\mu_=, \mu_{+-}\}$.

Proof. Clearly $\mu_* \geq \min\{\mu_=, \mu_{+-}\}$. For the other direction, assume w.l.o.g. $\alpha_1 \ldots, \alpha_m \in \mathcal{C}_-$, $\alpha_{m+1} \ldots, \alpha_k \in \mathcal{C}_+$ and
$\alpha_{m+k+1} \ldots, \alpha_n \in \mathcal{C}_=$. By Fact 2.2.2 and Proposition 4.4.3 we have that

$$\begin{aligned}
\mu_* &= \min\left\{\frac{\sum_{i=1}^{n} t_i \cdot w_i}{\sum_{i=1}^{n} t_i \cdot l_i}\right\} \\
&\geq \min\left\{\frac{\sum_{i=1}^{m+k} t_i \cdot w_i}{\sum_{i=1}^{m+k} t_i \cdot l_i}, \frac{\sum_{i=m+k+1}^{n} t_i \cdot w_i}{\sum_{i=m+k+1}^{n} t_i \cdot l_i}\right\} \\
&\geq \min\left\{\mu_{+-}, \mu_=\right\}
\end{aligned}$$

In section 4.5 we show how μ_* can be computed. Then, in section 4.6 we show that there exists words w_1, w_2 achieving μ_* and that no pair of words can achieve a value better than μ_*, i.e. that ω-NED(L_1, L_2) is indeed μ_*.

4.5 Computing μ_*

Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be complete NBAs for ω-regular languages L_1, L_2. We want to calculate ω-NED(L_1, L_2). We create $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$, the edit distance graph of the NBAs (from Definition 4.3.1). By Claim 4.2.4 a walk in $\mathcal{G}_{\text{ED}}$ represents an edit path from a word in L_1 to a word in L_2. For the path to be accepted by both NBAs it needs to visit an accepting state of $\mathcal{B}_i$ infinitely often for $i \in \{1, 2\}$. Thus we are interested in maximal strongly connected components (MSCC) that have at least one state from F_1 and at least one state from F_2. We can hence remove all vertices that do not reach such an MSCC. For simplicity we assume that our graph has one such MSCC (if this is not the

case, we apply our algorithm to each such MSCC separately).

We need the following construction to make sure pairs of cycles are balanced and that we evaluate all pairs of cycles:

Definition 4.5.1 (The balance t-counter graph) *Let $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2) = (V, L \times B, E, \theta)$ be the edit distance graph of the NBAs, and let $n = |V|$. Recall that $L = \Sigma_\varepsilon \times \Sigma_\varepsilon$ and $B = \{-1, 0, +1\}$. For threshold $t \in \mathbb{N}$ we define $\mathcal{G}^t_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$ as the labeled weighted graph $(V_t, L_t, E_t, \theta_t)$ where $V_t = V \times V \times [-t, t]$; the labeling function L_t omits the balance label from edges, i.e. $L_t = L$; the weight function θ_t associates with edge (n, l, n') for $n, n' \in V_t$ and $l \in L$ the weight $\theta_{\text{ED}}(l)$ (as in Definition 4.2.2); and the edges are $E_t = E'_t \cap V_t^2$ where*

$$\begin{aligned} E'_t \;=\; & \{(\langle u, v, i\rangle, l, \langle u', v, j\rangle) \;\Big|\; (u, \langle l, b\rangle, u') \in E, \quad b = j - i \;\} \\ \cup\; & \{(\langle u, v, i\rangle, l, \langle u, v', j\rangle) \;\Big|\; (v, \langle l, b\rangle, v') \in E, \quad b = j - i \;\} \end{aligned}$$

Let μ_t be the minimal mean simple cycle in $\mathcal{G}^t_{\text{ED}}$.

Lemma 4.5.2 *There exist $t \in \mathbb{N}$ such that $\mu_t = \mu_*$.*

Proof. Recall that by Corollary 4.4.4 $\mu_* = \min\{\mu_=, \mu_{+-}\}$. For the first direction ($\geq$) let

$$C = ((u_1, v_1, b_1), (u_2, v_2, b_2), \ldots, (u_r, v_r, b_r), (u_1, v_1, b_1))$$

be a simple cycle in $\mathcal{G}_t$ such that $\theta(C)/|C| = \mu_t$. We can project the path C on each coordinate we get two closed walks in $\mathcal{G}^t_{\text{ED}}$:
$C_1 = (u_1, \ldots, u'_r, u_1)$ and $C_2 = (v_1, \ldots, v''_r, v_1)$.
Since we have two closed walks we can decompose them into simple cycles $\alpha_1, \ldots \alpha_{m+k+l}$.

Now we can partition them into sets according to their balances. Let

$$\{\alpha_1, \ldots, \alpha_m\} \subseteq \mathcal{C}_-$$

$$\{\alpha_{m+1}, \ldots, \alpha_{m+k}\} \subseteq \mathcal{C}_+$$

$$\{\alpha_{m+k+1}, \ldots, \alpha_{m+k+l}\} \subseteq \mathcal{C}_=$$

and let $t_1, \ldots, t_{m+k+l}$ denote their respective number of repetitions in the path C. The claim now follows from Corollary 4.4.4.

For the second direction ($\leq$), we show that we can find paths corresponding to $\mu_=$ and μ_{+-} of $\mathcal{G}_{\text{ED}}$ in $\mathcal{G}^t_{\text{ED}}$.

Case of $\mu_=$: Let $c_=$ be a zero balanced simple cycle in $\mathcal{G}_{\text{ED}}$ achieving $\mu_=$. Assume

$$c_= \quad = \quad v_1 \xrightarrow{l_1,b_1} v_2 \xrightarrow{l_2,b_2} \cdots \xrightarrow{l_{k-1},b_{k-1}} v_k \xrightarrow{l_k,b_k} v_1$$

where $\sum_{i=1}^{k} b_i = 0$ and $\frac{\theta(c_=)}{|c_=|} = \mu_=$. We observe that for big enough t and for every $r \in [-n, n]$ the following cycle c^r_t is in $\mathcal{G}^t_{\text{ED}}$ and it achieves the same value.

$$\begin{aligned} c^r_t = (v_1, v_1, r) \quad &\xrightarrow{l_1} (v_2, v_1, r+b_1) \xrightarrow{l_2} \cdots \\ &\xrightarrow{l_{k-1}} (v_k, v_1, r+b_k) \xrightarrow{l_k} (v_1, v_1, r) \end{aligned}$$

Since $c_=$ is reachable from the initial state and the balance of a simple path is never larger than the length of the path there exists an $r \in [-n, n]$ such that (v_1, v_1, r) is reachable. Therefore, taking $t > n + n$ suffices.

Case of μ_{+-}: Now, let c_- and c_+ be a negatively and positively balanced simple

cycles, resp., in $\mathcal{G}_{\text{ED}}$ achieving μ_{+-}. Assume

$$c_- = u_1 \xrightarrow{l_1,b_1} u_2 \xrightarrow{l_2,b_2} \cdots \xrightarrow{l_{m-1},b_{m-1}} u_m \xrightarrow{l_m,b_m} u_1$$

and

$$c_+ = v_1 \xrightarrow{l'_1,b'_1} v_2 \xrightarrow{l'_2,b'_2} \cdots \xrightarrow{l'_{k-1},b'_{k-1}} v_k \xrightarrow{l'_k,b'_k} v_1$$

where $b_- = \sum_{i=1}^{m} b_i < 0$, $b_+ = \sum_{j=1}^{k} b'_j > 0$, $\frac{\theta(c_-)}{|c_-|} = \mu_-$ and $\frac{\theta(c_+)}{|c_+|} = \mu_+$.

In $\mathcal{G}^t_{\text{ED}}$ for big enough t we can find a cycle c^t_r corresponding to repeating b_+ times the cycle c_-, and then repeating b_- time the cycle c_+. It will have the following form

$$\begin{array}{llll}
(u_1, v_1, r_{1,1}), & (u_2, v_1, r_{1,2}), & \ldots, & (u_m, v_1, r_{1,m}), \\
(u_1, v_1, r_{2,1}), & (u_2, v_1, r_{2,2}), & \ldots, & (u_m, v_1, r_{2,m}), \\
 & & \ldots & \\
(u_1, v_1, r_{b_+,1}), & (u_2, v_1, r_{b_+,2}), & \ldots, & (u_m, v_1, r_{b_+,m}), \\
(u_1, v_1, r'_{1,1}), & (u_2, v_1, r'_{1,2}), & \ldots, & (u_k, v_1, r'_{1,k}), \\
(u_1, v_1, r'_{2,1}), & (u_1, v_2, r'_{2,2}), & \ldots, & (u_1, v_k, r'_{2,k}), \\
 & & \ldots & \\
(u_1, v_1, r'_{b_+,1}), & (u_1, v_2, r'_{b_+,2}), & \ldots, & (u_1, v_k, r'_{b_-,k}),
\end{array}$$

where if $r_1 = r$ then $r_{i,j} = r + i(b_-) + \sum_{k=1}^{j} b_i$ and $r'_{i,j} = r + (b_+)(b_-) + i(b_+)\sum_{k=1}^{j} b'_i$. Hence $r'_{b_-,k} = r + (b_+)(b_-) + (b_-)(b_+) = r$ and c^t_r is a balanced cycle of $\mathcal{G}^t_{\text{ED}}$. Since c_- is reachable from the initial state and the balance of a path is never larger than the length of the path there exists $r \in [-n, n]$ such that (u_1, v_1, r) is reachable. Since both $-b_-, b_+ < n$ we have that $t > n + n^2$ suffices. □

In fact a smaller bound exists, in the following lemma we improve the quadratic bound

to linear. The idea of the proof is that in $\mathcal{G}^t_{\text{ED}}$ it is possible to traverse part of the negative cycle, then move to traverse part of the positive cycle, and continue traversing the negative cycle from where we left of. Alternating between portions of the cycle we can make sure the balance is never more than n or less than $-n$.

Lemma 4.5.3 $\mu_{2n} = \mu_*$.

Proof. We use the same idea as in the previous proof, except instead of transitioning the first cycle and then the second cycle, we alternate between them. There exists minimal indices $i_0 = 0, i_1, i_2 \ldots$ such that $bal(c^\omega_+[i_{j-1}..i_j]) = \lceil \frac{n \cdot j}{2} \rceil$ for all $j > 0$. Similarly exists minimal indices $i'_1, i'_2 \ldots$ such that $bal(c^\omega_-[i'_{j-1}..i'_j]) = -\lceil \frac{n \cdot j}{2} \rceil$ for all $j > 0$. We alternate the positive by progressing in the positive cycle until we reach $\min\{i_j, -b_-|c_+|\}$ then we progress on the negative cycle until we reach $\min\{i'_j, b_+|c_-|\}$. We alternate between them until we reach a balance of 0 (after $-b_-$ of the positive cycle and b_+ of the negative cycle). Since both b_+ and $-b_-$ are at most n we have that at any given point our total balance will be in the range of $[-n, n]$. Therefore $t > n + n = 2n$ suffices. □

4.6 Showing ω-ned$(L_1, L_2) = \mu_*$

Next, we would like to show that $\omega\text{-NED}(L_1, L_2) = \mu_*$. Claim 4.6.1 shows that for every ultimately periodic words $w_1 \in L_1$ and $w_2 \in L_2$ we have $\omega\text{-NED}(w_1, w_2) \geq \mu_*$. Proposition 4.6.4 shows that this is the case also for arbitrary words $w_1 \in L_1$ and $w_2 \in L_2$.

On the other hand, we show that μ_* can be achieved by respective words $w_1 \in L_1$ and $w_2 \in L_2$. Claim 4.6.3 shows that we can get arbitrarily close to μ_* in the sense that for every $\varepsilon > 0$ we can find w_1, w_2 such that

$|\omega\text{-NED}(w_1, w_2) - \mu_*| < \varepsilon$.

Claim 4.6.1 (Lower-bound) *For any* $w_1, w_2 \in L_i \cap \Sigma^{\text{UP}}$, $\omega\text{-NED}(w_1, w_2) \geq \mu_*$.

Proof. Let (v_1, v_2) be a best rotation pair for the periods of w_1, w_2 and let $\rho \in \mathbb{D}^*$ be the optimal edit path for (v_1, v_2). We show that there exists a cycle c in $\mathcal{G}^t_{\text{ED}}$ that is accepting in both NBAs and satisfies $\theta(c)/|c| = \text{NED}(v_1, v_2)$.

For $i \in \{1, 2\}$ let $\beta_i \cdot \gamma_i^\omega$ be an accepting lasso run of $\mathcal{B}_i$ on w_i where γ_i reads v_i and assume w.l.o.g. $|\gamma_i| = |v_i|$. Assume $\beta_1 = q_1 q_2 \dots q_{\ell_1}$, $\beta_2 = q'_1 q'_2 \dots q'_{\ell_2}$, $\gamma_1 = p_1 p_2 \dots p_\ell$, $\gamma_2 = p'_1 p'_2 \dots p'_\ell$. Consider the paths $\beta = (q_1, q'_1), (q_2, q'_1), \dots, (q_{\ell_1}, q'_1), (q_{\ell_1}, q'_2), (q_{\ell_1}, q'_3) \dots (q_{\ell_1}, q'_{\ell_2})$ and $\gamma = (p_{i_0}, p'_{j_0}), \dots, (p_{i_k}, p'_{j_k})$ where

$$i_r = \begin{cases} 0 & r = 0 \\ i_{r-1} & \text{if } \rho[r] = d_{\text{E}} \text{ or } i_r = |\gamma_1| \\ i_{r-1}+1 & \text{otherwise} \end{cases} \qquad j_r = \begin{cases} 0 & r = 0 \\ j_{r-1} & \text{if } \rho[r] = d_{\text{S}} \text{ or } j_r = |\gamma_2| \\ j_{r-1}+1 & \text{otherwise} \end{cases}$$

Since $|\gamma_1| = |\gamma_2|$ we have that $|\rho| = |\gamma|$ and γ is a reachable cycle with value: $\theta(\gamma)/|\gamma| = wgt(\rho)/len(\rho) = cost(\rho)$. □

The proof of Claim 4.6.3 makes use of the following lemma.

Lemma 4.6.2 *[Cycle polarity] Let* $c = ((q_1, p_1), (q_2, p_2), \dots, (q_t, p_t), (q_1, p_1))$ *be a cycle in* $\mathcal{G}_{\text{ED}}$ *(where edge labels have been removed). We can decompose* c *into two cycles* c_+, c_- *in* $\mathcal{G}_{\text{ED}}$ *such that* $c_+ \in \mathcal{C}_+ \cup \mathcal{C}_=$ *and* $c_- \in \mathcal{C}_- \cup \mathcal{C}_=$.

Proof. Let $i_1, \ldots, i_k$, $j_1, \ldots, j_k$ be the indices where we took an edge from $\mathcal{B}_1, \mathcal{B}_2$, respectively. Then

$$c_+ = ((q_1, p_{i_1}), (q_1, p_{i_2}), \ldots, (q_1, p_{i_k}), (q_1, p_{i_1}))$$
$$c_- = ((q_{j_1}, p_1), (q_{j_2}, p_1), \ldots, (q_{j_k}, p_1), (q_{j_1}, p_1))$$

The balanced of c_+ may not be negative, since it has no delete edges, and that of c_- may not be positive, since it has no insert edges. □

Claim 4.6.3 (Upper-bound) *For all $\varepsilon > 0$ there exists $w_1^\omega \in L_1 \cap \Sigma^{\text{UP}}, w_2^\omega \in L_2 \cap \Sigma^{\text{UP}}$ such that $|\omega\text{-NED}(w_1^\omega, w_2^\omega) - \mu_*| \leq \varepsilon$.*

Proof. We show that both $\mu_=$ and μ_{+-} can be expanded to a valid edit path. Let $\varepsilon > 0$.
Case of $\mu_=$: Let $c_=$ be a simple cycle in $\mathcal{G}_{\text{ED}}$ with $val(c_=) = \mu_=$ and $bal(c_=) = 0$. By our assumption on the MSCC, there exists a cycle c traversing an accepting state from both F_1 and F_2. W.l.o.g. it intersects $c_=$. We claim that we can assume $bal(c) = 0$. Assume w.l.o.g. $bal(c) = b > 0$. We can find a negatively balanced cycle that starts from the same point as $c_=$ by tracing only edges corresponding to $\mathcal{B}_1$ and staying put on a fixed state of $\mathcal{B}_2$ (for details see Lemma 4.6.2). Then $c' = (c)^{(b-)}(c_-)^{(b)}$ is a cycle in $\mathcal{G}_{\text{ED}}$ and it is zero balanced.

Since $c_=$ is reachable from the initial state we have a path ρ_u from the initial state to $c_=$. Let ρ_v be the cycle $((c_=)^{n_\varepsilon} \cdot c)$. From here we create the walk $\rho = \rho_u \cdot (\rho_v)^\omega$.

Recall that a path in $\mathcal{G}_{\text{ED}}$ is an element of $(V \times L \times V)^\infty$ where $V = Q_1 \times Q_2$ and $L = \Sigma_\varepsilon \times \Sigma_\varepsilon$. Given a path ρ in $\mathcal{G}_{\text{ED}}$, we use $labels(\rho)$ for $\pi_2(\rho) \in (\Sigma_\varepsilon \times \Sigma_\varepsilon)^\infty$. For $i \in \{1, 2\}$, let $u_i = \pi_i(labels(\rho_u))$, $v_i = \pi_i(labels(\rho_v))$ and $w_i = u_i(v_i)^\omega$. Then $w_i \in L_i$ since the corresponding runs of $\mathcal{B}_i$ visits an accepting state in F_i infinitely often.

We turn to show that $\omega\text{-NED}(w_1, w_2) - \mu_= < \varepsilon$. Recall that $\omega\text{-NED}(w_1, w_2) = \text{NED}(v_1, v_2)$ following Theorem 3.2. Note that since ρ_v is balanced $|v_1| = |v_2|$. For large enough n_ε:

$$\begin{aligned}
\omega\text{-NED}(w_1, w_2) - \mu_= &= \text{NED}(v_1, v_2) - \mu_= \\
&\leq \frac{\theta(\rho_v)}{|\rho_v|} - \mu_= \\
&= \frac{n_\varepsilon \cdot \theta(c_=) + \theta(c)}{n_\varepsilon \cdot |c_=| + |c|} - \frac{\theta(c_=)}{|c_=|} \\
&\leq \frac{\theta(c)}{n_\varepsilon \cdot |c_=|} < \varepsilon
\end{aligned}$$

Case of μ_{+-}: In a similar fashion we have the simple cycles $c_+, c_- \; \mathcal{G}_{\text{ED}}$ with $val(c_+) = \mu_+$, $val(c_-) = \mu_-$ and $bal(c_+) = b_+ > 0, bal(c_-) = b_- < 0$. Since c_+, c_- are in the same SCC, we have a cycle $c = p_1 p_2$ where p_1 is from the first state of c_+ to the first state of c_- and p_2 comes back to the first state in c_+ while containing accepting states from both F_1 and F_2. We can assume that $bal(c) = b$ is 0 since if this is not the case we can use Lemma 4.6.2 to extract from c a positive and a negative cycle, and achieve a balance of zero by repeating each the number of times that corresponds to the balance of the other.

Since c_+ is reachable from the initial state we have a path ρ_u from the initial state to c_+. Let ρ_v be the cycle

$$(c_+)^{-b_- \cdot n_\varepsilon} \cdot p_1 \cdot (c_-)^{b_+ \cdot n_\varepsilon} \cdot p_2.$$

□

For $i \in \{1, 2\}$, let $u_i = \pi_i(labels(\rho_u))$, $v_i = \pi_i(labels(\rho_v))$ and $w_i = u_i(v_i)^\omega$. Then $w_i \in L_i$ since the corresponding runs of $\mathcal{B}_i$ visits an accepting state in F_i infinitely

often. Thus, for large enough n_ε,

$$\begin{aligned}
&\omega\text{-NED}(w_1, w_2) - \mu_{+-} = \\
&\quad= \omega\text{-NED}(v_1^\omega, v_2^\omega) - \mu_{+-} \\
&\quad\leq \frac{\theta(\rho_v)}{|\rho_v|} - \mu_{+-} \\
&\quad= \frac{n_\varepsilon{\cdot}(-b_-{\cdot}\theta(c_+){+}b_+{\cdot}\theta(c_-)){+}\theta(c)}{n_\varepsilon{\cdot}(-b_-{\cdot}|c_+|{+}b_+{\cdot}|c_-|){+}|c|} - \frac{-b_-{\cdot}|c_+|{+}b_+{\cdot}|c_-|}{-b_-{\cdot}|c_+|){+}b_+{\cdot}|c_-|} \\
&\quad\leq \frac{\theta(c)}{n_\varepsilon{\cdot}(-b_-{\cdot}|c_+|{+}b_+{\cdot}|c_-|} < \varepsilon
\end{aligned}$$

Proposition 4.6.4 (Ultimately periodic words suffice)

$$\omega\text{-NED}(L_1, L_2) = \omega\text{-NED}(L_1 \cap \Sigma^{\text{UP}}, L_2 \cap \Sigma^{\text{UP}})$$

Proof. Clearly $\omega\text{-NED}(L_1, L_2) \leq \omega\text{-NED}(L_1 \cap \Sigma^{\text{UP}}, L_2 \cap \Sigma^{\text{UP}})$. For the other direction, it suffices to show that for all $\varepsilon > 0$ and any $w_1 \in L_1, w_2 \in L_2$ there exists $w_1' \in L_1 \cap \Sigma^{\text{UP}}, w_2' \in L_2 \cap \Sigma^{\text{UP}}$ such that $|\omega\text{-NED}(w_1', w_2') - d| < \varepsilon$ where $d = \omega\text{-NED}(w_1, w_2)$. Let r_1, r_2 be accepting runs of w_1, w_2 in their respective NBAs, $\mathcal{B}_1$ and $\mathcal{B}_2$. Let q_1 and q_2 be such that $r_1[i] = q_1$ and $r_2[i] = q_2$ for infinitely many is. Such a pair exists by the pigeonhole principle. Let n_* be the first index such that $r_1[n_*] = q_1$ and $r_2[n_*] = q_2$.

Since the number of states in both automata is finite, there is a $C \in O(|\mathcal{G}_{\text{ED}}|^2)$ such that if there is a balanced path between some pair of states in $\mathcal{G}_{\text{ED}}$ then there is one that is shorter than C. Let $n_0 > \max\{n_*,\ 2C/\varepsilon\}$ be such that $|\text{NED}(w_1[..n_0], w_2[..n_0]) - d| < \varepsilon/2$. Let ρ be the corresponding edit path. Let $n_{**} \geq n_0$ be the first index after n_0 such that $r_1[n_{**}] = q_1$ and $r_2[n_{**}] = q_2$ and accepting states of both NBAs have been visited along the way. Let $u_i = w_i[..n_*]$, $v_i = w_i[n_*{+}1..n_0]$. Let ρ_z be a path

from $(r_1[n_0], r_2[n_0])$ to $(r_1[n_{**}], r_2[n_{**}])$ and z_i the words obtained by projecting ρ_z on the NBAs $\mathcal{B}_i$ for $i \in \{1, 2\}$. Note that the length of ρ_z is bounded by C and thus also its weight. Let ρ_v be an optimal edit path for v_1, v_2. Let i be the first such that both coordinates of $\mathit{endpoint}(\rho[..i])$ are greater or equal to n_* (see Figure 4.1). Assume w.l.o.g. that $\mathit{endpoint}(\rho[..i]) = (n_*, n_*{+}\delta)$ where $\delta \geq 0$. Because (n_*, n_*) is

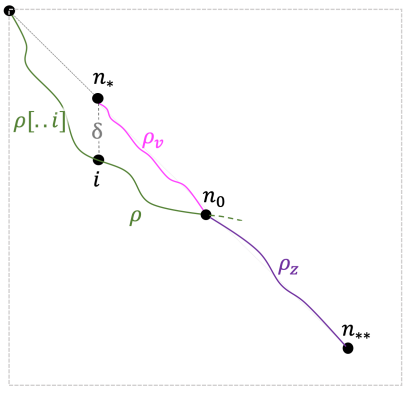


Figure 4.1: Auxiliary Figure for the proof of Proposition 4.6.4

on the diagonal of $\mathcal{G}(v_1 z_1, v_2 z_2,)$ the number of south edges in ρ is at least δ, thus $wgt(\rho[..i]) \geq \delta$. Also $wgt(\rho_v) \leq \delta + wgt(\rho[i{+}1..n_0])$ and similarly for the weights.

Further $n_* + \delta \geq len(\rho[..i]) \geq wgt(\rho[i..])$. Thus

$$\begin{aligned}
\omega\text{-NED}\,(u_1(v_1z_1)^\omega, u_2(v_2z_2)^\omega) - d \quad &\leq \quad \text{NED}(v_1z_1, v_2z_2) - d \quad \leq \\
&\leq \quad \frac{wgt(\rho_v) + wgt(\rho_z)}{len(\rho_v) + len(\rho_z)} - d \\
&\leq \quad \frac{\delta + wgt(\rho[i{+}1..n_0]) + wgt(\rho_z)}{\delta + len(\rho[i{+}1..n_0]) + len(\rho_z)} - d \\
&\leq \quad \frac{n_* + \delta + wgt(\rho[i{+}1..n_0]) + wgt(\rho_z)}{n_* + \delta + len(\rho[i{+}1..n_0]) + len(\rho_z)} - d \\
&\leq \quad \frac{wgt(\rho[..i]) + wgt(\rho[i{+}1..n_0]) + wgt(\rho_z)}{len(\rho[..i]) + len(\rho[i{+}1..n_0]) + len(\rho_z)} - d \\
&\leq \quad \frac{wgt(\rho) + wgt(\rho_z)}{len(\rho) + len(\rho_z)} - d \\
&\leq \quad \frac{wgt(\rho) + C}{len(\rho)} - d \quad = \quad \frac{wgt(\rho)}{len(\rho)} - d + \frac{C}{n_0} \leq \varepsilon
\end{aligned}$$

The first inequality follows from Theorem 3.2. We use inequality because the concerned periods may not be a best rotation. The second follows since $\rho \cdot \rho_z$ is an edit path for (v_1z_1, v_2z_2). For the third inequality, recall that (n_*, n_*) is on the diagonal of $\mathcal{G}(v_1z_1, v_2z_2,)$ and Because the number of south edges in ρ is at least δ, $wgt(\rho[..i]) \geq \delta$. The third inequality follows since the optimal path from (n_*, n_*) to (n_{**}, n_{**}) is better than going δ steps to the south and then following ρ to (n_0, n_0) and then following ρ_z. The rest follows by applying arithmetic on our assumptions and noticing that if $\hat{\rho}$ is an edit path for w_1, w_2 then $cost(\hat{\rho}) \geq cost(\rho)$ by Fact 2.2.2. □

Corollary 4.6.5 $\omega\text{-NED}(L_1, L_2) = \mu_*$

Remark 4.6.6 (Distance on Parity automata) *We claim that the same idea can be used to compute the distance between two regular ω-languages given by non-deterministic parity automata $\mathcal{P}_1$ and $\mathcal{P}_2$. We work with the definition saying that a run is accepting if the minimum color visited infinitely often is even. Consider $\mathcal{G}_{\text{ED}}(\mathcal{P}_1, \mathcal{P}_2)$ constructed*

as in Definition 4.3.1. A zero balanced path in $\mathcal{G}_{\text{ED}}$ that corresponds to accepting runs of $\mathcal{P}_1$ and $\mathcal{P}_2$ will traverse some MSCC and the minumum color visited in $\mathcal{P}_i$ for $i \in \{1,2\}$ will be even. We can thus consider MSCCs where the minimum color from $\mathcal{P}_i$ for $i \in \{1,2\}$ is even. To this aim we can simply remove states where the color in component i (for $i \in \{1,2\}$) corresponds to the minimum for i in the MSCC and it is odd, and repeat this process until we are left with MSCCs satisfying this criterion. Then we can compute $\mu_* = \min\{\mu_=, \mu_{+-}\}$ in the same manner. To see that there are words w_1, w_2 corresponding to μ_* we apply the same idea as in Claim 4.6.3 but instead of extending the cycle to visit a state of the accepting state F_i of the NBA, we extend the cycle to visit the minimum color of both $\mathcal{P}_1$ and $\mathcal{P}_2$. Since the process of extracting the MSCCs of interest can also be done in PTIME *(it requires at most the number of colors times computation of MSCCs) we conclude that the ω-*NED *distance between two languages given by parity automata can also be done in* PTIME.

Remark 4.6.7 (Distance on Muller automata) *We further claim that the idea can also be used when the languages are given by non-deterministic Muller automata $\mathcal{M}_1$ and $\mathcal{M}_2$. Assume the acceptance condition of $\mathcal{M}_i$ is $\mathcal{F}_i = \{F_1, \ldots, F_{k_i}\}$ for $i \in \{1,2\}$. Now it suffices to consider only SCCs (not necessarily maximal) whose projection on $\mathcal{M}_i$ is in $\mathcal{F}_i$ for $i \in \{1,2\}$. There are $k_1 \times k_2$ such SCCs. For each such SCC, we can compute μ_*. Take an SCC corresponding to some F_i in $\mathcal{F}_i$ for $i \in \{1,2\}$. To show words w_1, w_2 achieving μ_* we extend the path by visiting each state of F_1 and each state of F_2. This adds a factor of $|Q_1| \times |Q_2|$. Thus, the ω-*NED *distance between two languages given by Muller automata can also be done in* PTIME.

4.7 The algorithm

For clarity, we finish off this chapter by describing the algorithm to compute ω-NED between languages. Given L_1, L_2 ω-regular languages and their corresponding Büchi automata, $\mathcal{B}_1 = (\Sigma, Q_1, s_1, \delta_1, F_1), \mathcal{B}_2 = (\Sigma, Q_2, s_2, \delta_2, F_2)$ we provide the full algorithm for computing $\omega\text{-NED}(L_1, L_2)$.

Algorithm 4.1: ω-NED

Input: $\mathcal{B}_1 = (\Sigma, Q_1, s_1, \delta_1, F_1)$, $\mathcal{B}_2 = (\Sigma, Q_2, s_2, \delta_2, F_2)$, t - threshold

output: $\omega\text{-NED}(L_1, L_2) = \min_{w_1 \in L_1, w_2 \in L_2} \omega\text{-NED}(w_1, w_2) = \mu_*$

initialize: $G := \mathcal{G}^t_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$ - the balance t-counter graph Definition 4.5.1 , $\mu_* := 1$

for *MSCC* M *in* G *such that* $M \cap (F_1 \times Q_2) \cap (Q_1 \times F_2) \neq \emptyset$ **do**
- $\mu \leftarrow$ minimal mean value amongst simple cycles of G (using [Kar78])
- $\mu_* \leftarrow \min(\mu_*, \mu)$

return μ_*

Example 4.7.1 *Consider the languages* $L_1 = \{aaab\}^\omega, L_2 = \{aab, ab\}^\omega$. *They can be represented by the NBAs* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *given in Figure 4.3.*

Running the algorithm on these two NBAs, we get the result $\frac{2}{9}$. *We can see that indeed the edit-path in Figure 4.2 is of cost* $\frac{2}{9}$ *and it corresponds to the words* $aaabaaab \in L_1$ *and* $aababaab \in L_2$.

In this example, $t = 1$ *suffices since we have an optimal cycle whose balance index never goes above* 1 *or below* 0. *Figure 4.4 shows this cycle; one can observe that the counter is always within* 0 *and* 1. *Figure 4.5 shows the respective positive and negative cycles.*

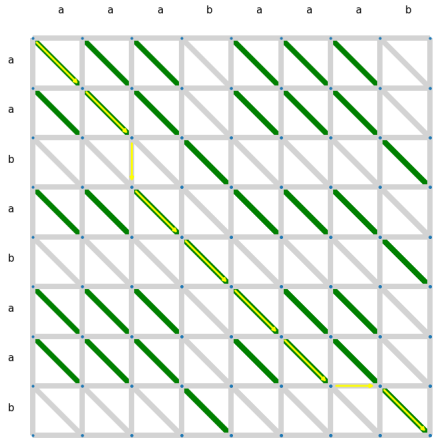


Figure 4.2: South-East graph with edit-path showing ω-NED$((aaab)^\omega, (aababaab)^\omega) =$ NED$(aaabaaab, aababaab) = \frac{2}{9}$.

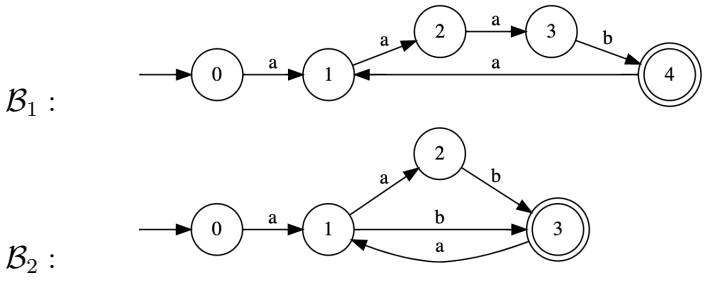


Figure 4.3: An NBA $\mathcal{B}_1$ for $L_1 = \{aaab\}^\omega$ and an NBA $\mathcal{B}_2$ for $L_2 = \{aab, ab\}^\omega$ as inputs for section 4.7.

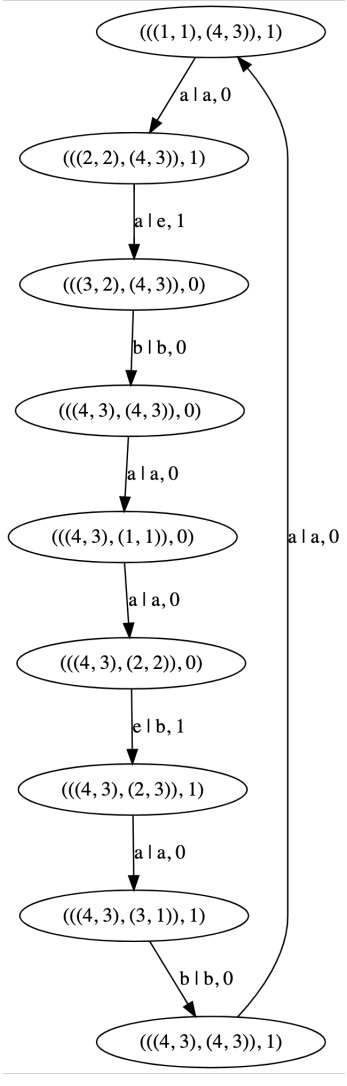


Figure 4.4: Minimal cycle of $\mathcal{G}^t_{\mathrm{ED}}(\mathcal{B}_1, \mathcal{B}_2)$

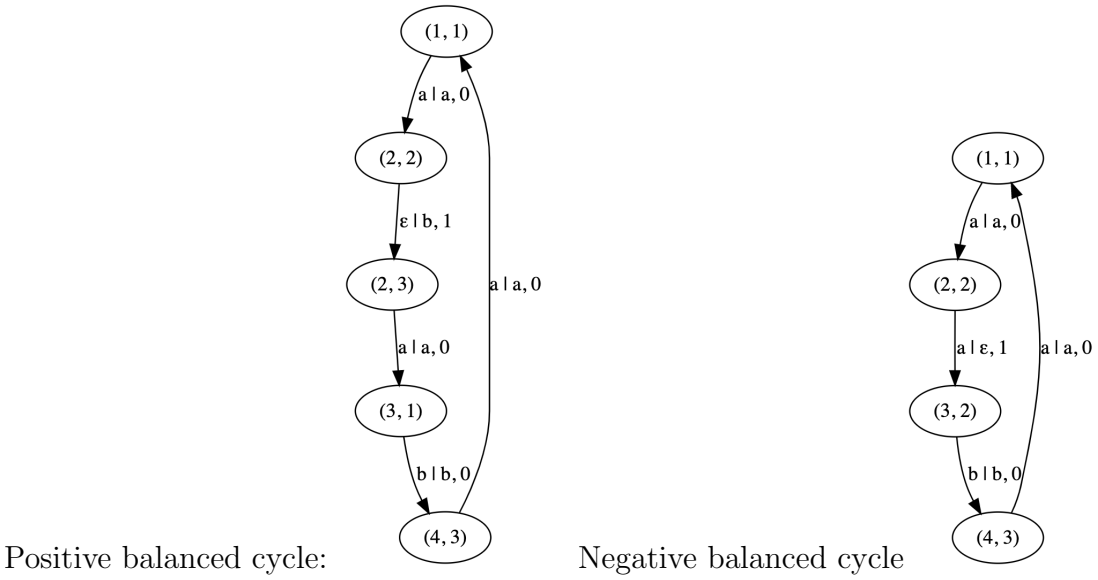


Figure 4.5: Original cycles in $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$

Chapter 5

Reflecting on the transient part

We have managed to provide an edit distance for infinite words that answers the desired criteria. In particular, this measure was designed to ignore differences that occur in finite prefixes of the words. However, if the given two words are ultimately equal, e.g. if $x_1 = a^4ba^\omega$ and $x_2 = a^4ca^\omega$ it would be nice to get a measure that reflects that there is some difference on the transient parts, and perhaps more informatively, that the average number of edit operations required in the transient part of the words is $1/5$.

To crystallize the desired intuition from such an edit distance function we review Tabuada and Neider's definition of *robust LTL*, in short rLTL. The idea is to refine the truth values of *satisfies* and *violates* (or *true* and *false*) to capture, in the case the formula is violated, how severe is the violation. In rLTL, the answer to whether a given infinite word satisfies a formula is one of five values $\{0, 1, 2, 3, 4\}$. The intuition can be understood by considering the following five ultimately periodic words and the formula

$\mathbf{G}p$ (stating that p should always hold).

$$w_0 = (p)^\omega, \quad w_1 = \overline{p}^{10} \cdot (p)^\omega, \qquad w_2 = (\overline{p}^{10} \cdot p)^\omega,$$
$$w_3 = \overline{p}^{10} \cdot p \cdot (\overline{p})^\omega, \quad w_4 = (\overline{p})^\omega$$

With regard to the question does w_i satisfy $\mathbf{G}p$ in rLTL the answer is $4-i$. That is, the answers are decreasing from the best score 4 to the lowest score 0. Note that while in the current work we measure distance, and hence 0 is the optimal value, Tabuada and Neider measure satisfaction, thus the higher the better (and 4 is the highest). Let's take a closer look to understand [TN16]'s scores. The word w_0 gets the perfect score 4 since it satisfies the formula at hand (since p indeed always holds). The word w_1 gets the second best score 3 since while p doesn't always hold it almost always holds (i.e. it holds in all but finitely many times). The word w_2 gets the score 2 since p no longer almost always holds, but it does hold infinitely often. The score for w_3 and w_4 goes down to 1 and 0, respectively, since p does not hold infinitely often, but at least for w_3 there is some time point where p holds.

We find this scoring system appealing and thus aim to find an edit distance function for ultimately periodic words that refines it. In particular, when computing the distance between the words w_i and p^ω we expect the result to reflect the proportion of p's in the transient and periodic parts of the word. That is, consider $w_1' = p^{50}\,\overline{p}\,^{50}p^\omega$ and $w_1'' = p^{80}\,\overline{p}\,^{20}p^\omega$. According to [TN16] the satisfaction score of both with respect to $\mathbf{G}p$ is 3; but we expect the distance between w_1' and $w = p^\omega$ to be greater than the distance between w_1'' and w since more edit operations are required to get from w_1' to w than from w_1'' to w. For the same reason we expect the difference between $w_2' = (\overline{p}^{70} \cdot p^{30})^\omega$ to w to be greater than the distance between $w_2'' = (\overline{p}^{10} \cdot p^{90})^\omega$ and w.

In order to obtain a distance function d_{tp} that reflects the distances in both the transient part and the periodic part, as we wish, we can simply combine a distance function d_t that reflects distances in the transient part and a distance function d_p that reflects distances in the periodic part in a pair and define the order of pairs using lexicographic order (henceforth *lex order*) Using Fact 2.2.3 we can almost reduce the problem to finding suitable measures to the transient and periodic parts. For instance we can define d_{tp} based on ω-NED for d_p and $Disc_\lambda$ for d_t. Such a definition would give us the desired intuition of the normalized number of edits required in the period, but as mentioned already in the introduction, $Disc_\lambda$ cannot model the normalized number of edits in the transient part. If the transient part is given to us explicitly, we can instead use distance measures for finite words and apply it on the transient part, using the following Fact 2.2.4.

The obvious candidate to apply to the transient part is of course NED. This gives rise to the following definition.[1]

Definition 5.0.1 *Let* $u_1, u_2 \in \Sigma^*$ *and* $v_1, v_2 \in \Sigma^+$. *Let* $w_1 = u_1(v_1)^\omega$ *and* $w_2 = u_2(v_2)^\omega$. *We define*

$$\text{UP\$NED}(\langle u_1, v_1\rangle, \langle u_2, v_2\rangle) = (\text{NED}(u_1, u_2), \omega\text{-NED}(w_1, w_2))$$

We can immediately conclude that

Theorem 5.2 $(\Sigma^* \times (\Sigma^{\text{UP}}/\equiv)), \text{UP\$NED})$ *is a metric space.*

[1] In this definition and the following ones the left component corresponds to the transient part and the right corresponds to the periodic part. Usually, we view the periodic part as more important, hence the lexicographic order we use would consider the right component as the one with higher significance. We prefer to write the transient part on the left since it is on the left of an ultimately periodic word (which is read left to write).

This definition answers the desired intuition for the pairs of words discussed above, but it also gives that

$$\text{UP}_\$\text{NED}(\langle aaaa, a\rangle, \langle a, a\rangle) = (\tfrac{3}{4}, 0)$$

while both $\langle aaaa, a\rangle$ and $\langle a, a\rangle$ represent the same ultimately periodic word a^ω. In other words, it doesn't preserve the property that the distance between different representations of the same ultimately periodic word is zero. Formally,

Property 5.0.3 *If $u_1(v_1)^\omega = u_2(v^2)^\omega$ Then $d_{tp}(\langle u_1, v_1\rangle, \langle u_2, v_2\rangle) = (0, 0)$.*

However, it can make sense in applications where the length of the transient part reflects something inherent like the length of an initialization sequence.

If a separation between the transient and the periodic parts is not given to us a priory, then we cannot use such a measure without deciding where the separation between the transient and periodic parts should be placed. We can choose the separation point to be the least index i such that $w[i..]$ is periodic, we denote this distance $\text{UP}_\bullet\text{NED}$ and define it formally below. Another option is to assume some $k \in \mathbb{N}$ is given and consider for the transient part the prefix $w[..k]$.

Definition 5.0.4 *Let $w_1, w_2 \in \Sigma^\omega$ be two ultimately periodic words. We define*

$$\text{UP}_\bullet\text{NED}(w_1, w_2) \stackrel{\text{def}}{=\!=} (\text{NED}(u_1', u_2'), \omega\text{-NED}(w_1, w_2))$$
$$\text{UP}_k\text{NED}(w_1, w_2) \stackrel{\text{def}}{=\!=} (\text{NED}(w_1[..k], w_2[..k]), \omega\text{-NED}(w_1, w_2))$$

where for $i \in \{1, 2\}$, u_i' is the shortest prefix of w_i such that there exists v_i' for which $w_i = u_i'(v_i')^\omega$.

Property 5.0.3 is clearly preserved by both $\text{UP}_\bullet\text{NED}$ and UP_kNED because if $w_1 =$

$u_1(v_1)^\omega$, $w_2 = u_2(v_2)^\omega$ and $w_1 = w_2$ then $u'_1 = u'_2$ and $w_1[..k] = w_2[..k]$. Since the transient part is not explicitly given, we cannot simply apply Fact 2.2.3 or Fact 2.2.4 to deduce these two definitions are metrics. However, we can still show that they are.

Theorem 5.5 $(\Sigma^{\text{UP}}/\equiv, \text{UP}_\bullet\text{NED})$ *is a metric space.*

Proof. For identity of indiscernibles, note that the second component of $\text{UP}_\bullet\text{NED}$ is different than zero iff $w_1 \not\equiv w_2$. If $w_1 \equiv w_2$ yet $w_1 \neq w_2$ then w_1 and w_2 differ in the transient part. Thus $u'_1 \neq u'_2$ implying the first component of $\text{UP}_\bullet\text{NED}$ will be different than zero. And if the first component is not zero then clearly $w_1 \neq w_2$. Symmetry clearly holds. Triangle inequality holds since the u'_i part is determined given w_i, and hence the reasoning of Fact 2.2.4 applies. □

Considering UP_kNED we define two ultimately periodic words as equivalent, denoted $w_1 \equiv_k w_2$ iff $w_1 \equiv w_2$ and $w_1[..k] = w_2[..k]$.

Theorem 5.6 $(\Sigma^{\text{UP}}/\equiv_k, \text{UP}_k\text{NED})$ *is a metric space.*

Proof. Again, the second component of UP_kNED is different than zero iff $w_1 \not\equiv w_2$. The first component is different than zero iff $w_1[..k] \neq w_2[..k]$. Thus $\text{UP}_k\text{NED}(w_1, w_2)$ is different than zero iff $w_1 \not\equiv_k w_2$, thus identity of indiscernibles holds. Symmetry clearly holds. Triangle inequality holds since $w[..k]$ is deterministically determined given w, and hence the reasoning of Fact 2.2.4 applies. □

Let us discuss the results provided by $\text{UP}_\bullet\text{NED}$ and UP_kNED for various examples. Consider the words $x_1 = (b^4a)c^\omega$ and $x_2 = (ab^4)c^\omega$. We have that $\text{UP}_\bullet\text{NED}(x_1, x_2) = (\frac{2}{6}, 0)$ and $\text{UP}_k\text{NED}(x_1, x_2) = (\frac{2}{k}, 0)$ for $k \geq 5$. Both are plausible. Consider $x'_1 = (b^4a)c^\omega$ and $x'_2 = (ab^4)d^\omega$. We have that $\text{UP}_\bullet\text{NED}(x'_1, x'_2) = (\frac{2}{6}, 1)$ and $\text{UP}_k\text{NED}(x'_1, x'_2) = (\frac{k-3}{k+1}, 1)$

for $k \geq 5$. The latter is not plausible for large k's as it blurs the difference in the transient part. In general, for words which are not almost equal, for large enough k's the first component of UP_kNED may not reflect well the differences in the transient part.

Consider now $y_1 = b^\omega$, $y_2 = a^3b^\omega$ and $y_3 = a^{100}b^\omega$. We get that $\text{UP}_\bullet\text{NED}(y_1, y_2) = \text{UP}_\bullet\text{NED}(y_1, y_3) = (1, 0)$ which is not satisfactory since it blurs that y_2 requires less edit operations to get to y_1 then y_3. For these words we get that $\text{UP}_k\text{NED}(y_1, y_2) = (\frac{3}{k}, 0)$ for $k \geq 3$ and $\text{UP}_k\text{NED}(y_1, y_3) = (\frac{100}{k}, 0)$ for $k \geq 100$ which is more plausible.

Another interesting example concerns (fully) periodic words whose periods are permutations one of the other, e.g. $z_1 = (abc)^\omega$ and $z_2 = (bca)^\omega$. We get that $\text{UP}_\bullet\text{NED}(z_1, z_2) = (0, 0)$ whereas $\text{UP}_k\text{NED}(z_1, z_2) = (\frac{2}{k}, 0)$ for $k \geq 2$, so the former does not reflect that there is some difference in the initial part, while the latter does.

To summarize, while we have provided three normalized edit distance functions for ultimately periodic words, none meets the desired intuition on *all* examples. So it seems that the choice of which one to choose should depend on the application. Or perhaps better ones can be suggested in future work.

Part II

Normalized edit distance on finite words

Chapter 6

The Normalized Edit Distance with Uniform Operation Costs is a Metric

The proof that ω-NED is a metric relys on the *till now* open question whether NED is a metric. This chapter closes this long standing gap in the literature. We first introduce different notation used in the paper, and present the proof that NED is a metric. Finally we compare NED to other known edit distances and describe properties that show some of the benifits of using NED.

6.1 Representation of edit paths

When describing ω-NED we focused on calculating the value of ω-NED, therefore the *south-east graph* and *endpoint* let us prove our calculations were correct. The main

challenge in proving NED is a metric is showing that it obeys the triangle inequality. Since in this proof we need to construct a new path given 2 paths, we introduce new notations that allow us to easily construct such paths using a strict set of rules. For convenience in this proof we use a slightly different representation of edit paths, using edit letters instead of the *south-east graph* graph defined in Part I.

Basic 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.

Extended edit letters 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 $\mathtt{n}$ 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 introduce new edit letters whose length is different than 1. Thus, the more general definition of *len*.

Example 6.1.1 *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 6.1.2 *We define a function apply*: $\Sigma^* \times \Gamma_\Sigma^* \to \Sigma^* \cup \{\perp\}$ *that given a string* w *over* Σ*, and an edit path* p *over* Γ_Σ *obtains a new string* w' *over* Σ *as follows. It returns* $\perp$ *if the edit path is invalid for the input word.*

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

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 6.1.3 *Following on Example 6.1.1, 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.*

6.2 Normalized edit distance definition

The normalized edit distance Let p be an edit path. The *cost* of p, denoted $cost(p)$

$$cost(p) = \begin{cases} \frac{wgt(p)}{len(p)} & \text{if } |p| > 0 \\ 0 & \text{otherwise} \end{cases}$$

Using the definition of *cost* we can define the notion we study in this chapter, namely the *normalized edit distance*, NED, of Marzal and Vidal [MV93].

Definition 6.2.1 (The normalized edit distance, ned [MV93]) *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 6.2.2 *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 changes 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* $\text{NED}(s_1, s_2) = \frac{3}{4}$.

6.3 Proof of metric

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 6.3.1 *Let $s, s_1, s_2 \in \Sigma^*$. Then*

1. $\text{NED}(s, s) = 0$

2. *if* $s_1 \neq s_2$ *then* $\text{NED}(s_1, s_2) > 0$

3. $\text{NED}(s_1, s_2) = \text{NED}(s_2, s_1)$

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 $\overline{p_{12}}$ 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 $\text{NED}(s_1, s_2) = \text{NED}(s_2, s_1)$. □

The challenge is proving that NED satisfies the third condition, the triangle inequality.

6.3.1 A Proof of the Triangle Inequality

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

$$d_{13} \leq d_{12} + d_{23} \tag{6.3.1}$$

$$l_{13} \geq \max\{l_{12}, l_{23}\} \tag{6.3.2}$$

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 `b` (for *blank*). This is actually an abbreviation of `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 `b` symbols. The method $cmps_h \colon \Gamma_\Sigma^* \times \Gamma_\Sigma^* \to (\Gamma_\Sigma \cup \{\mathtt{b}\})^* \cup \{\perp\}$ is defined inductively, in Definition 6.3.2, 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

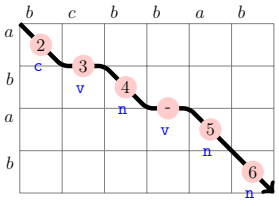
s_2 and s_2 into s_3, respectively, $cmps_h$ is well defined.

Definition 6.3.2 *Let p_{12}, p_{23} be edit paths over Γ_Σ. We define $cmps_h(p_{12}, p_{23})$ inductively as follows.*

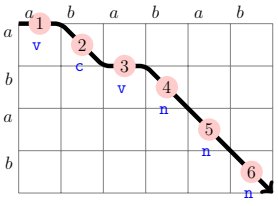
$$cmps_h(p_{12}, p_{23}) = \begin{cases} \varepsilon & \text{if } p_{12} = p_{23} = \varepsilon & (0) \\ \mathtt{x}_\sigma \cdot cmps_h(p_{12}[2..], p_{23}) & \text{if } p_{12}[1] = \mathtt{x}_\sigma & (1) \\ \mathtt{v}_\sigma \cdot cmps_h(p_{12}, p_{23}[2..]) & \text{if } p_{23}[1] = \mathtt{v}_\sigma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

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 $\mathtt{b}$ results in the string s_3. Figure 6.1 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 6.3.1) and (Equation 6.3.2) hold.

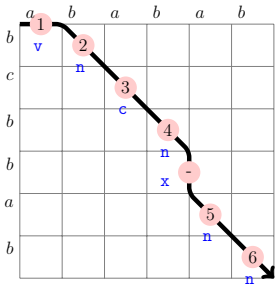
Note that if we reach case (12) then we cannot claim that the result is an edit path. We thus first show that if $cmps_h$ is applied to two edit paths p_{12}, p_{23} such



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



(b) The composed edit path p_{13} using Definition 6.3.2



(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 6.1: Let $w_1 = abab$, $w_2 = bcbbbab$, $w_3 = abababab$. Figure 6.1(a) shows an optimal edit path p_{12} between w_1 to w_2, Figure 6.1(c) shows an optimal edit path p_{23} between w_2 to w_3. Figure 6.1(b) shows the edit path p_{13} composed from p_{12} and p_{23} using Definition 6.3.2. The edit operations in Figure 6.1(b) are marked with numbers 1 to 6. A number n in between 1 and 6 in Figure 6.1(a) and Figure 6.1(c) signifies that the corresponding edge contributed to the construction of the edge marked n in Figure 6.1(b) (thus for the operations corresponding to cases (1) and (2) of Definition 6.3.2, there is one corresponding marking in Figure 6.1(a) and Figure 6.1(c) and for the others there are two). The labels $-$ in Figure 6.1(a) and Figure 6.1(c) 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}.

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 6.3.3 *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 $p_{12}[1] = \mathtt{x}_\sigma$. 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 `x` and the first character of p_{23} is not `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 $s_2[2..] = apply(p_{12}[2..], s_1)$.

   - **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.

   - **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.

   - **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 $s_2[2..] = apply(p_{12}[2..], s_1[2..])$.

- **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.

- **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.

- **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 $s_2[2..] = apply(p_{12}[2..], s_1[2..])$.

   - **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.

   - **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.

- **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 6.3.1 and Equation 6.3.2. 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. Our claim is that it is better than going through s_2.

Proposition 6.3.4 *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. $d_{13} \leq d_{12} + d_{23}$
3. $l_{13} \geq \max\{l_{12}, l_{23}\}$

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 $d_{13} = 0 \leq d_{12} + d_{23} = 0$ and $l_{13} = 0 \geq \max\{l_{12}, l_{23}\} = 0$

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 Lemma 6.3.3 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 Definition 6.3.2.

(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. $d'_{13} \leq d'_{12} + d_{23}$
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. $d_{13} = 1 + d'_{13} \leq 1 + d'_{12} + d_{23} = d_{12} + d_{23}$
3. $l_{13} = 1 + l'_{13} \geq 1 + \max\{l'_{12}, l_{23}\} \geq \max\{1 + l'_{12}, l_{23}\} = \max\{l_{12}, l_{23}\}$.

(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. $d'_{13} \leq d_{12} + d'_{23}$
3. $l'_{13} \geq \max\{l_{12}, l'_{23}\}$.

Therefore

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

(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. $l'_{13} \geq \max\{l'_{12}, l'_{23}\}$.

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. $d_{13} = 1 + d'_{13} \leq 1 + d'_{12} + d'_{23} = d_{12} + d_{23}$
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. $l_{13} = 2 + l'_{13} > 1 + \max\{l'_{12}, l'_{23}\} = \max\{1 + l'_{12}, 1 + l'_{23}\} = \max\{l_{12}, l_{23}\}$.

□

Recall that *cost* is defined as *wgt* divided by *len*. Let p_{13} be the string obtained by compose in Proposition 6.3.4. Then by items 2 and 3 we know that

$$wgt(p_{13}) \leq wgt(p_{12}) + wgt(p_{23}) \tag{6.3.3}$$

$$len(p_{13}) \geq \max\{len(p_{12}), len(p_{23})\} \tag{6.3.4}$$

We can thus conclude from Lemma 6.3.10 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 6.3.5 *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 Lemma 6.3.9, we conclude that $\frac{wgt(p_{13})}{len(p_{13})} \geq \frac{wgt(h(p_{13}))}{len(h(p_{13}))}$.

Corollary 6.3.6 *$cost(p) \geq cost(h(p))$*

Proposition 6.3.7 *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 Corollary 6.3.5 we get that $cost(p_{13}) \leq cost(p_{12}) + cost(p_{23})$. From Proposition 6.3.4 it holds that $h(p_{13})$ is a valid edit path over Γ_Σ. From Corollary 6.3.6 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 6.8 *The Normalized Levenshtein Distance* NED *(provided in Definition 6.2.1) 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 Proposition 6.3.1. The third condition, namely triangle inequality, follows from Proposition 6.3.7. □

6.3.2 Properties of fractions

Lemma 6.3.9 *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 6.3.10 *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. $\frac{d_{13}}{l_{13}} \leq \frac{d_{12}+d_{23}}{l_{13}} = \frac{d_{12}}{l_{13}} + \frac{d_{23}}{l_{13}} \leq \frac{d_{12}}{l_{12}} + \frac{d_{23}}{l_{23}}$. □

6.4 Properties of the various normalized edit distance functions

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[MV93, dlHM08]. The main contribution of the chapter 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 6.4.1 *In Example 6.2.2 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 Example 6.1.1, 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.

6.4.1 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 *wgt* 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 [Lev66].

Definition 6.4.2 (The edit (Levenstein) distance, ED) *The* edit distance *between s_i and s_j, denoted $\text{ED}(s_i, s_j)$, is the minimal weight of a path p_{ij} from s_i to s_j. That is,*

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

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

The *post-normalized edit distance* divides the weight by the sum of the length of the words. Likewise one can consider definitions which use the min or max of the lengths. These three definitions were introduced just to show they are not a metric [MV93, dlHM08].

Definition 6.4.3 (ped_{sum}, ped_{min}, ped_{max}) *Let s_i and s_j be strings.*

$$\text{PED}_{sum}(s_i, s_j) = \min \left\{ \frac{wgt(p_{ij})}{|s_i| + |s_j|} \;\middle|\; apply(p_{ij}, s_i) = s_j \right\}$$

$$\text{PED}_{min}(s_i, s_j) = \min \left\{ \frac{wgt(p_{ij})}{\min(|s_i|, |s_j|)} \;\middle|\; apply(p_{ij}, s_i) = s_j \right\}$$

$$\text{PED}_{max}(s_i, s_j) = \min \left\{ \frac{wgt(p_{ij})}{\max(|s_i|, |s_j|)} \;\middle|\; apply(p_{ij}, s_i) = s_j \right\}$$

It was shown in [MV93] that PED_{sum} does not satisfy the triangle inequality, and in [dlHM08] that PED_{min} and PED_{max} do not satisfy the triangle inequality either.

Example 6.4.4 *Let* $\Sigma = \{a, b, c, d\}$.

- *Let* $s_1 = aa$, $s_2 = ab$ *and* $s_3 = b$. *Using the uniform weights, we have that* $\text{PED}_{sum}(s_1, s_2) = \frac{1}{4}$, $\text{PED}_{sum}(s_2, s_3) = \frac{1}{3}$, $\text{PED}_{sum}(s_1, s_3) = \frac{2}{3}$ *but* $\frac{1}{4} + \frac{1}{3} = \frac{7}{12} < \frac{2}{3}$. *Thus, the triangle inequality does not always hold.*

- *Let* $s_1 = abc$, $s_2 = abcd$ *and* $s_3 = bdd$. *Using the uniform weights, we have that* $\text{PED}_{min}(s_1, s_2) = \frac{1}{3}$, $\text{PED}_{min}(s_2, s_3) = \frac{2}{3}$, $\text{PED}_{min}(s_1, s_3) = \frac{4}{3}$ *but* $\frac{1}{3} + \frac{2}{3} = 1 < \frac{4}{3}$. *Thus, the triangle inequality does not always hold.*

- *Let* $s_1 = ab$, $s_2 = aba$ *and* $s_3 = ba$. *Using the uniform weights, we have that* $\text{PED}_{max}(s_1, s_2) = \frac{1}{3}$, $\text{PED}_{max}(s_2, s_3) = \frac{1}{3}$, $\text{PED}_{max}(s_1, s_3) = \frac{2}{2} = 1$ *but* $\frac{1}{3} + \frac{1}{3} = \frac{2}{3} < 1$. *Thus, the triangle inequality does not always hold.*

We turn to introduce the *generalized normalized edit distance* proposed and proven to be a metric by Li and Liu [LL07].

Definition 6.4.5 (The generalized edit distance) $\text{GED}(s_i, s_j) = \frac{2 \cdot \text{ED}(s_i, s_j)}{|s_i| + |s_j| + \text{ED}(s_i, s_j)}$.

Last, we define of the *contextual edit distance*, proposed and proven to be a metric by de la Higuera and Micó [dlHM08]. 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 6.4.6 (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,*

$$\text{CED}(s_x, s_y) = \min \left\{ \text{CED}(\alpha) \;\middle|\; \alpha = (s_0, s_1, \ldots, s_k),\ s_0 = s_x,\ s_k = s_y,\ \text{ED}(s_i, s_{i+1}) = 1 \right\}$$

6.4.2 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 y_1 and y_2 to be less than the distances between the x_1 and x_2 but this is not the case for ED, as can be observed by the following examples.

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

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 6.4.7 *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. □

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 $\text{CED}'(s_1, s_2) = \min(1, \text{CED}(s_1, s_2))$. [1]

Property 6.4.8 (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.[2]

Claim 6.4.9 *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.[3] 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;

[1] This is inspired by [Lit19] 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"*.

[2] 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 $\text{CED}(\varepsilon, a) = 1 < \infty$.

[3] 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.

which holds iff there are no edit letters with weight zero. Since the only zero weight edit letter is swap of identical letters, the value of NED is 1 if and only if the words have no common letter. □

It is note worthy to notice that GED takes into consideration difference in sizes (the addition of length of words) while NED only takes into consideration the length of the edit-path (which is between the maximal word and the sum of both words)

$$\begin{aligned} &\text{GED}(a, b^{100}) = \tfrac{200}{201} && \text{NED}(a, b^{100}) = 1 \\ &\text{GED}(a^i, b^j) = \tfrac{2\cdot\max\{i,j\}}{i+j+\max\{i,j\}} && \text{NED}(a^i, b^j) = 1 \end{aligned}$$

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 6.4.10 (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 6.4.11 *The* NED *and* GED *distances satisfy the property of non escalation of repetitions. The* CED *and* CED' *distances do not.*

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

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

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:

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

The same reasoning shows that GED does not escalate repetitions.

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

□

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 6.4.12 (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*

$$d(s_1, s_2) = \min \{ d(s'_1, s'_2) \mid s'_i \in (\Sigma \uplus \Sigma_i)^* \text{ and } \pi_\Sigma(s'_i) = s_i \text{ for } i \in \{1, 2\} \}.$$

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

Claim 6.4.13 *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 6.4.12 clearly holds. For the $\leq$ direction, we turn to Claim 6.4.14 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 6.4.12, 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. □

Claim 6.4.14 *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 Lemma 6.3.9 that $cost(p) \leq cost(p')$ □

Chapter 7

Discussion and Conclusions

This thesis presents a natural extension of the normalized edit distance (denoted NED) from finite words to infinite words (which we denote ω-NED) and shows that this extension is a metric. To the best of our knowledge this is the first edit distance over infinite words known to be a metric. We show that the ω-NED distance between two ultimately periodic words can be computed in PTIME. We further investigate how to compute the ω-distance between two ω-regular languages. We provide algorithms that answer this question, when the languages are represented using (non-deterministic) Büchi, Parity or Muller automata, models that are commonly used in verification.

In the process of proving ω-NED is a metric, we came across an open question — whether the well know normalized edit distance NED used with uniform weights is a metric (on finite words). We closed this gap by providing a proof that indeed it satisfies the triangle inequality.

For future work we think it would be interesting to check if our linear t bound (Lemma 4.5.3) for which $\mu_t = \mu_*$ is optimal, and if not, find a tighter bound. We would like to inves-

tigate computational complexity of ω-NED with regard to other computational models (such as pushdown automata).

The notion of distance between languages has been rigorously studied in the formal methods community, in particular as means for answering questions related to quality of implementations, robustness, and repair (see [CHR10, BPR11, CHR12, CHOV17, NWZ19, FMR+20] to name a few). Most of these works define the distance between languages in a different way than we do. While we define $d(L_1, L_2)$ as $\inf_{w_1 \in L_1, w_2 \in L_2} d(w_1, w_2)$ these works define $d(L_1, L_2)$ as $\sup_{w_1 \in L_1} \inf_{w_2 \in L_2} d(w_1, w_2)$. Our definition follows the standard way to extend a definition of a metric from elements to sets. Their definition, which is not a metric (for instance since it is not symmetric) builds on other motivation, it looks for the worst-case number of operations needed to get from L_1 to L_2. For future work we would like to consider the various other questions regarding distances in formal verification (e.g. the *threshold distance*, the *correctness distance*, the *robustness distance*), and devise algorithms for solving them for ω-NED.

Bibliography

[Ber79] Jean Berstel. *Transductions and context-free languages*, volume 38 of *Teubner Studienbücher : Informatik*. Teubner, 1979.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BPR11] Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Regular repair of specifications. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 335–344, 2011.

[CDH10] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010.

[CGK+18] Edmund M. Clarke, Orna Grumberg, Daniel Kroening, Doron A. Peled, and Helmut Veith. *Model checking, 2nd Edition*. MIT Press, 2018.

[CHOV17] Krishnendu Chatterjee, Thomas A. Henzinger, Jan Otop, and Yaron Velner. Quantitative fair simulation games. *Inf. Comput.*, 254:143–166, 2017.

[CHR10] Pavol Cerný, Thomas A. Henzinger, and Arjun Radhakrishna. Quantitative simulation games. In Zohar Manna and Doron A. Peled, editors,

Time for Verification, Essays in Memory of Amir Pnueli, volume 6200 of *Lecture Notes in Computer Science*, pages 42–60. Springer, 2010.

[CHR12] Pavol Cerný, Thomas A. Henzinger, and Arjun Radhakrishna. Simulation distances. *Theor. Comput. Sci.*, 413(1):21–35, 2012.

[DG08] Volker Diekert and Paul Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, pages 261–306, 2008.

[dlHM08] 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.

[EF06] Cindy Eisner and Dana Fisman. *A Practical Introduction to PSL*. Series on Integrated Circuits and Systems. Springer, 2006.

[FGMW22] Dana Fisman, Joshua Grogin, Oded Margalit, and Gera Weiss. The normalized edit distance with uniform operation costs is a metric, 2022. submitted to CPM, the 33rd Annual Symposium on Combinatorial Pattern Matching.

[FGW22] Dana Fisman, Joshua Grogin, and Gera Weiss. A normalized edit distance on infinite words, 2022. submitted to LICS, the Thirty-Seventh Annual ACM/IEEE Symposium on Logic in Computer Science.

[FMR+20] Emmanuel Filiot, Nicolas Mazzocchi, Jean-François Raskin, Sriram Sankaranarayanan, and Ashutosh Trivedi. Weighted transducers for ro-

bustness verification. In *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, pages 17:1–17:21, 2020.

[HP84] David Harel and Amir Pnueli. On the development of reactive systems. In Krzysztof R. Apt, editor, *Logics and Models of Concurrent Systems - Conference proceedings, Colle-sur-Loup (near Nice), France, 8-19 October 1984*, volume 13 of *NATO ASI Series*, pages 477–498. Springer, 1984.

[HR86] H. J. Hoogeboom and G. Rozenberg. *Infinitary languages: Basic theory and applications to concurrent systems*, pages 266–342. Springer Berlin Heidelberg, Berlin, Heidelberg, 1986.

[Joh98] Richard Johnsonbaugh. A Discrete Intermediate Value Theorem. `https://www.maa.org/sites/default/files/0746834259610.di020780.02p0372v.pdf`, 1998. The College Mathematical Journal.

[Kar78] Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discret. Math.*, 23(3):309–311, 1978.

[Lev66] Vladimir Iosifovich Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, feb 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[Lit19] Christopher C. Little. `https://abydos.readthedocs.io/en/latest/abydos.distance.html#abydos.distance.HigueraMico`, 2014-2019.

[LL07] Yujian Li and Bi Liu. A normalized levenshtein distance metric. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1091–1095, 2007.

[MV93] Andrés Marzal and Enrique Vidal. Computation of normalized edit distance and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):926–932, 1993.

[NWZ19] Daniel Neider, Alexander Weinert, and Martin Zimmermann. Robust, expressive, and quantitative linear temporal logics: Pick any two for free. In *Proceedings Tenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2019, Bordeaux, France, 2-3rd September 2019*, pages 1–16, 2019.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[TN16] Paulo Tabuada and Daniel Neider. Robust linear temporal logic. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, pages 10:1–10:21, 2016.

[Var95] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In Faron Moller and Graham M. Birtwistle, editors, *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, Banff, Canada, August 27 - September 3, 1995, Proceedings)*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1995.