# A Similarity Measure for Formal Languages Based on Convergent Geometric Series

Florian Bruse, Maurice Herwig, and Martin Lange

University of Kassel, Kassel, Germany

**Abstract.** We present a distance metric on formal languages based on the accumulated weight of words in their symmetric difference. The contribution of an individual word to this weight decreases exponentially in its length, guaranteeing the distance between languages to be a real value between 0 and 1. We show that this distance is computable for regular languages. As an application, we show how the similarity measure derived from a modification of this metric can be used in automatic grading of particular standard exercises in formal language theory classes.

## 1 Introduction

A similarity measure is a function that associates a numerical value to two objects of some class in order to quantify how similar they are. Such a measure typically satisfies certain properties, for instance symmetry, since $A$ should be as similar to $B$ as $B$ is to $A$. It should be monotonic in the sense that the numerical similarity of $A$ and $B$ should be higher than that of $A$ and $C$, when $C$ appears to be more different to $A$ than $B$ does. The triangular inequality ensures that the numerical values measure similarity evenly throughout the entire class: $C$ cannot be appear to be less similar to $A$ than the sum of the values indicating the similarity of $C$ to $B$ and that of $B$ to $A$. Finally, it should satisfy a maximality condition ensuring that no $B$ can be more similar to $A$ than $A$ itself.

There is not an established theory of similarity measures, probably because of its close connection to the established theory of *distance* in metric spaces. In fact, similarity can be seen as the absence of distance in such a space. In other words, one can typically obtain a similarity measure satisfying the properties above by inverting a distance metric on such objects.

Similarity measures are heavily used in many applications, not at all restricted to formal language theory or even computer science. They play vital roles in psychology [3] and educational sciences [11] but also in other fields within computer science, for example image processing [9], bioinformatics [13] and data science [5]. In fact, it is the recent resurgence of machine learning and the progress it has brought to many applications areas which has created new interest in the study of similarity measures. Supervised learning relies on a thorough understanding of similarity between objects, for example images, in order to guarantee reliable classification results by trained neural networks.

Fundamental formal language theory provides results and concepts which may apply to other areas through the encoding of their primary objects of study

by words and languages, cf. modelling of DNA sequences or program runs as words over a finite alphabet. Similarity measures and, more specifically, distance metrics have therefore been studied in the context of formal language theory, mainly on words. Essentially all known types of distance/similarity like Euclidian, Manhattan, Cosine, Hamming, Levenshtein, Jaccard, etc. can be used to turn the set of words over some alphabet into a metric space.

A distance measure $d$ on words naturally induces a distance $\hat{d}$ on languages via $\hat{d}(L_1, L_2) := \min\{d(w_1, w_2) \mid w_i \in L_i\}$. This is unsuitable as a basis for a similarity measure on languages for many applications, though, as it ignores the inner structure of these languages. Worst of all, it considers two languages to be closest, and therefore most similar, already when their intersection is nonempty. For example, $\{a^n b^n \mid n \geq 1\}$ and $a^*b^*$ would be as similar as $a^*b^*$ is to itself. A more involved definition lifts these distance notions from words to languages using the Hausdorff distance, i.e. $\hat{d}(L_1, L_2) = \max\big(\{\tilde{d}(L_1, w) \mid w \in L_2\} \cup \{\tilde{d}(L_2, w') \mid w' \in L_1\}\big)$ where $\tilde{d}(L, w) = \min\{d(w', w) \mid w' \in L\}$. This notion assigns distance 0 only to equal sets, yet one quickly runs into undecidability problems using this definition [4].

In this paper we propose a new distance metric for formal languages, based on the sparsity of the symmetric difference between two languages. This is standard; however, given that the symmetric difference can be of infinite size, there is no standard way to obtain a finite numerical distance value from this. We define the distance as the accumulated weight of all words in the symmetric difference, where the weight of a word is exponential in the inverse of its length. This guarantees well-definedness of the distance metric, as it is bounded from above by a convergent geometric series.

In Sect. 2 we recall necessary preliminaries, present the formal definition of this distance metric, discuss its most important algebraic and computational properties, and compare it to other proposals from the literature. In Sect. 3 we show its usefulness in a particular application scenario: the similarity measure derived from this distance metric can be used to automatically assess and grade students' solutions to exercises of the form "construct an NFA / DFA / regular expression for the language $L = \ldots$." We show how this measure can be modified to better suit the needs of this application scenario, describe its implementation and how an empirical evaluation can be used to tweak the measure's parameters. Sect. 4 concludes with remarks on further work.

## 2 Distance Based on a Convergent Geometric Series

### 2.1 Formal Languages and Metrics

We briefly recall a few preliminaries about formal languages, finite automata and metric spaces.

An *alphabet* is a finite nonempty set $\Sigma = \{a, b, \ldots\}$ of *letters*. A (finite) $\Sigma$-*word* is a sequence $w = a_1 \cdots a_n$ of letters. Its length $|w|$ is $n$. The empty word is denoted by $\varepsilon$ and has length 0. $\Sigma^*$ is the set of all $\Sigma$-words; $\Sigma^n$ denotes the set

of $\Sigma$-words of length $n$. A *$\Sigma$-language* is an $L \subseteq \Sigma^*$. We write $L^{(n)}$ for $L \cap \Sigma^n$, $L^{(\leq n)}$ for $\bigcup_{i \leq n} L^{(i)}$ and $L^{(>n)}$ for $\bigcup_{i>n} L^{(i)}$. As usual, for $a \in \Sigma$ and $L \subseteq \Sigma^*$ we write $aL$ for the language $\{aw \mid w \in L\}$.

A *nondeterministic finite automaton* (NFA) is an $\mathcal{A} = (Q, \Sigma, \delta, Q_I, Q_F)$ where $Q$ is a finite non-empty set of *states*, $\Sigma$ is an alphabet, $\delta\colon Q \times \Sigma \to 2^Q$ is the *transition relation* and $Q_I, Q_F \subseteq Q$ are the sets of *initial*, resp. *final* states.

The extended transition relation $\hat{\delta}\colon Q \times \Sigma^* \to 2^Q$ can be obtained via $\hat{\delta}(q, \varepsilon) = \{q\}$ and $\hat{\delta}(q, wa) = \bigcup_{q' \in \hat{\delta}(q,w)} \delta(q', a)$. As usual $L(\mathcal{A}) := \{w \mid \text{ex. } q \in Q_I \text{ s.t. } \hat{\delta}(q, w) \cap Q_F \neq \emptyset\}$ is the language of words accepted by $\mathcal{A}$. An $L \subseteq \Sigma^*$ is *regular* if it is accepted by some NFA.

An NFA is called *deterministic*, or a DFA, if $|\delta(q, a)| = 1$ for all $q \in Q, a \in \Sigma$ and $|Q_I| = 1$. It follows by induction that $|\hat{\delta}(q, w)| = 1$ for all $q \in Q, w \in \Sigma$ and that the automaton accepts iff $\hat{\delta}(q, w) \in Q_F$ for the unique $q \in Q_I$. For an underlying DFA we simply write $\hat{\delta}(q, w) = q'$ instead of $\hat{\delta}(q, w) = \{q'\}$, etc.

It is folklore that for every NFA there is a DFA that accepts the same language, and that the regular languages are closed under the usual operations, in particular union, intersection, complementation, Kleene star, etc. [10].

Finally, we recall the definition of a *metric space*, i.e. a pair $(M, d)$ of a set $M$ and a function $d\colon M^2 \to \mathbb{R}^{\geq 0}$ satisfying the following properties for all $x, y, z \in M$:

- $d(x, y) = 0$ iff $x = y$ (identity of indiscernibles),
- $d(x, y) = d(y, x)$ (symmetry),
- $d(x, z) \leq d(x, y) + d(y, z)$ (the triangular inequality).

### 2.2 Formal Definition of the Distance Metric

Let $\Sigma$ be an alphabet. The distance metric presented in the following is defined as $d(L_1, L_2) = \omega_\lambda(L_1 \mathbin{\Delta} L_2)$ where $L_1 \mathbin{\Delta} L_2$ is the symmetric difference of $L_1$ and $L_2$ and $\omega_\lambda\colon 2^{\Sigma^*} \to [0, 1]$ is a sum-based weight function that assigns a real number to each language, depending on a parameter $\lambda \in \mathbb{Q}$ with $0 < \lambda < 1$.

Let $L \subseteq \Sigma^*$ be a language. Let $f_L\colon \mathbb{N} \to \mathbb{Q} \cap [0, 1]$ be defined as

$$f_L(n) = \frac{|L^{(n)}|}{|\Sigma^n|}$$

i.e. $f_L$ assigns to each number the fraction of all words in $\Sigma^n$ that are in $L$. The value of $\omega_\lambda(L)$ depends on the values of $f_L(n)$ for all $n \in \mathbb{N}$ and, hence, all of $L$. In order to derive a finite value from the generally infinite sequence $(f_L(n))_{n \in \mathbb{N}}$, we use the fact that the geometric series $\sum_{i=0}^{\infty} \lambda^i$ converges to $\frac{1}{1-\lambda}$ for all $\lambda \in [0, 1)$. Hence, if we discount the elements of the sequence $(f_L(n))_{n \in \mathbb{N}}$ by the corresponding terms of the geometric series, we obtain a finite value, i.e. for all $\lambda \in [0, 1)$, the infinite sum $\sum_{i=0}^{\infty} \lambda^i \cdot f_L(i)$ is well-defined and yields a value between 0 and $\frac{1}{1-\lambda}$. Clearly, this value is only interesting when $\lambda > 0$. Boundedness of the infinite sum follows from the fact that $\sum_{i=0}^{\infty} \lambda^i$ converges for $\lambda \in [0, 1)$ and $0 \leq f_L(i) \leq 1$ for all $i$.

In order to use the above for a similarity measure on languages that has a fixed range, we normalise the value obtained in the infinite sum to $[0,1]$. This yields, for each $\lambda \in (0,1)$, the following definition for a function $\omega_\lambda \colon 2^{\Sigma^*} \to [0,1]$.

**Definition 1.** *Let $\lambda \in (0,1)$ and let $\Sigma$ be an alphabet. Then the sum-based measure $\omega_\lambda$ is defined via*

$$\omega_\lambda(L) = (1-\lambda) \cdot \sum_{i=0}^{\infty} \lambda^i \cdot f_L(i) = (1-\lambda) \cdot \sum_{i=0}^{\infty} \lambda^i \cdot \frac{|L^{(i)}|}{|\Sigma^i|}.$$

We drop the parameter $\lambda$ if it is clear from context or not important, and simply write $\omega(L)$. We will use both the characterisation using $f_L(i)$ and that using the fraction, depending on which presentation is more useful in the situation.

**Observation 1.** For all $\lambda \in (0,1)$, we have that $\omega_\lambda(L) = 0$ iff $L = \emptyset$ and $\omega_\lambda(L) = 1$ iff $L = \Sigma^*$. Moreover, $\omega_\lambda$ is strictly monotone, i.e. if $L \subsetneq L'$ then $\omega_\lambda(L) < \omega_\lambda(L')$. The converse, is not true: $\omega_\lambda(L) < \omega_\lambda(L')$ does not imply $L \subsetneq L'$ as can easily be seen from the example where $L = \{\varepsilon, a\}$ and $L' = \{a, aa\}$ over the alphabet $\{a\}$. We have $\omega_{0.5}(L) = \frac{3}{4} > \frac{3}{8} = \omega_{0.5}(L')$.

Finally, if $L = L_1 \dot\cup L_2$, then $\omega_\lambda(L) = \omega_\lambda(L_1) + \omega_\lambda(L_2)$.

From this observation, we conclude that $\omega_\lambda$ is well-defined:

**Lemma 1.** *Let $\Sigma$ be an alphabet and let $\lambda \in (0,1)$ and let the sum-based function $d_\lambda \colon 2^{\Sigma^*} \times 2^{\Sigma^*} \to [0,1]$ be defined via $d_\lambda(L_1, L_2) = \omega_\lambda(L_1 \mathbin{\Delta} L_2)$. Then $d_\lambda$ is a metric on the space of all $\Sigma$-languages.*

*Proof.* By Obs. 1, $d_\lambda(L_1, L_2) = 0$ iff $L_1 = L_2$. Next, the definition of $d_\lambda$ is clearly inherently symmetric since $L_1 \mathbin{\Delta} L_2 = L_2 \mathbin{\Delta} L_1$. Hence, it only remains to show that $d_\lambda$ satisfies the triangular inequality.

Let $L_1, L_2, L_3 \subseteq \Sigma^*$. We have to show that $d_\lambda(L_1, L_2) + d_\lambda(L_2, L_3) \geq d_\lambda(L_1, L_3)$. Since $d_\lambda(L_1, L_3)$ is defined as $\omega_\lambda(L_1 \mathbin{\Delta} L_3)$ and since $\omega_\lambda$ is monotone by Obs. 1, it is sufficient to observe that if $w \in L_1 \mathbin{\Delta} L_3$ then $w \in L_1 \mathbin{\Delta} L_2$ or $w \in L_2 \mathbin{\Delta} L_3$ whence $L_1 \mathbin{\Delta} L_3$ can be divided into $L_1 \mathbin{\Delta} L_3 = D_1 \cup D_2$ such that $D_1 \subseteq L_1 \mathbin{\Delta} L_2$ and $D_2 \subseteq L_2 \mathbin{\Delta} L_3$. □

We now show that $\omega_\lambda(L)$ is computable if $L$ is regular and, hence, $d_\lambda$ constitutes a computable distance metric on the regular languages. Since the class of regular languages is closed under all Boolean operations – in particular symmetric differences – and distances are invariant under the formalism used to represent a language, it suffices to show that, for a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_i, Q_F)$, the value $\omega_\lambda(L(\mathcal{A}))$ is computable.

For $q \in Q$, let $L_q$ be the language defined by the automaton $\mathcal{A}_q = (Q, \Sigma, \delta, q, Q_F)$, i.e. the DFA that results from $\mathcal{A}$ by making $q$ the unique starting state. It is well-known that the languages $L_q$ can be characterised recursively via $L_q = \chi_q \cup \bigcup_{a \in \Sigma} aL_{\delta(q,a)}$ with $\chi_q = \{\varepsilon\}$ if $q \in Q_F$ and $\chi_q = \emptyset$ otherwise. We can use

a similar characterisation for the computation of the values of $\omega_\lambda(L_q)$ for all $q$. First, by definition we have

$$\omega_\lambda(L_q) = (1-\lambda) \cdot \sum_{i=0}^{\infty} \lambda^i \cdot \frac{|L_q^{(i)}|}{|\Sigma^i|} \ .$$

The individual terms of this infinite sum are all computable, for example $\frac{|L_q^{(0)}|}{|\Sigma^0|}$ is 1 if $L_q$ contains the empty word, and 0 otherwise. For $i \geq 1$, a word in $L_q^{(i)}$ is of the form $aw$ where $w \in L_{\delta(q,a)}^{(i-1)}$. Hence, the fraction $\frac{|L_q^{(i)}|}{|\Sigma^i|}$ can be rewritten as

$$\frac{|L_q^{(i)}|}{|\Sigma^i|} \ = \ \frac{|\bigcup_{a\in\Sigma} \ aL_{\delta(q,a)}^{(i-1)}|}{|\Sigma^i|} \ = \ \frac{\sum_{a\in\Sigma} \ |L_{\delta(q,a)}^{(i-1)}|}{|\Sigma^i|} \ . \tag{1}$$

It follows that the infinite sum for $\omega_\lambda(L_q)$ can also be rewritten as

$$\begin{aligned}
\omega_\lambda(L_q) &= (1-\lambda)\Big(|L_q^{(0)}| + \sum_{i=1}^{\infty} \lambda^i \cdot \frac{|L_q^{(i)}|}{|\Sigma^i|}\Big) \\
&= (1-\lambda) \cdot |L_q^{(0)}| + (1-\lambda) \cdot \sum_{i=1}^{\infty} \lambda^i \cdot \frac{\sum_{a\in\Sigma} |L_{\delta(q,a)}^{(i-1)}|}{|\Sigma^i|} \\
&= (1-\lambda) \cdot |L_q^{(0)}| + \lambda \cdot (1-\lambda) \cdot \sum_{i=1}^{\infty} \lambda^{i-1} \cdot \frac{\sum_{a\in\Sigma} |L_{\delta(q,a)}^{(i-1)}|}{|\Sigma| \cdot |\Sigma^{i-1}|} \\
&= (1-\lambda) \cdot |L_q^{(0)}| + \frac{\lambda}{|\Sigma|} \cdot \sum_{a\in\Sigma} (1-\lambda) \cdot \sum_{i=0}^{\infty} \lambda^i \cdot \frac{|L_{\delta(q,a)}^{(i)}|}{|\Sigma^i|} \\
&= (1-\lambda) \cdot |L_q^{(0)}| + \frac{\lambda}{|\Sigma|} \cdot \sum_{a\in\Sigma} \omega_\lambda(L_{\delta(q,a)})
\end{aligned}$$

We write $t_{q,q'}$ for $\frac{|\{a\in\Sigma|\delta(q,a)=q'\}|}{|\Sigma|}$ and $e_q$ for $|L_q^{(0)}|$, i.e. $e_q = 1$ if $q \in Q_F$ and $e_q = 0$ otherwise. Assume that $Q = \{q_1, \ldots, q_n\}$ for some $n$. Then the above equation can be rewritten as

$$\omega_\lambda(L_q) = (1-\lambda) \cdot e_q + \lambda \cdot t_{q,q_1} \omega_\lambda(L_{q_1}) + \cdots + \lambda \cdot t_{q,q_n} \omega_\lambda(L_{q_n}). \tag{2}$$

From this we derive the following equation system relating the values of $\omega_\lambda(L_q)$:

$$\begin{array}{lll}
-(1-\lambda) \cdot e_{q_1} = (\lambda \cdot t_{q_1,q_1} - 1) \cdot \omega_\lambda(L_{q_1}) & + \cdots + & \lambda \cdot t_{q_1,q_n} \cdot \omega_\lambda(L_{q_n}) \\
\quad \vdots \quad \vdots & \ddots & \vdots \\
-(1-\lambda) \cdot e_{q_n} = \lambda \cdot t_{q_n,q_1} \cdot \omega_\lambda(L_{q_1}) & + \cdots + & (\lambda \cdot t_{q_n,q_n} - 1) \cdot \omega_\lambda(L_{q_n})
\end{array} \tag{3}$$

By the above, this system is satisfied by the individual $L_q$ and their weights. Uniqueness of a solution can be shown in a standard way.

**Lemma 2.** *Let $\mathcal{A}$ be a DFA and let $\mathcal{E}$ be the set of equations associated to it as in Eq. 3. Then $\mathcal{E}$ possesses a unique solution.*

It follows that $\omega_\lambda(L)$ is computable if $L$ is regular.

**Theorem 2.** *Let $\mathcal{A}$ be a DFA with $n$ states. Then $\omega_\lambda(L(\mathcal{A}))$ can be computed in time $\mathcal{O}(n^3)$.*

The given runtime follows from the well-known complexity of e.g. Gaussian elimination. This does of course not preclude asymptotically better procedures based on other equation solvers.

It is a standard exercise to construct a DFA for the symmetric difference from two given DFA via the product construction. This then yields computability of both the weight function $\omega_\lambda$ and the distance metric $d_\lambda$ on regular languages.

**Corollary 1.** *Let $L_1, L_2$ be regular and given as a DFAs with $n$, resp. $m$ states. Then $d_\lambda(L_1, L_2)$ can be computed in time $\mathcal{O}((n \cdot m)^3)$.*

Clearly, computability holds also for any other representation that can be translated into DFA like NFA, (extended) regular expressions, formulas of Monadic Second-Order Logic, etc. with corresponding effect on the overall runtime. A fair question here concerns the feasibility of this approach for NFA directly without prior determinisation. The problem is that in an NFA a word $aw$ accepted from some state $q$ may have more than one accepting run. In particular, there may be different successors $q', q''$ s.t. $w \in L_{q'} \cap L_{q''}$. Since the weight of a language accumulates weights of all words in it, the approach sketched above would either count $w$ several times and thus include $aw$ into $L_q$ with a wrong weight; or there would have to be some mechanism that intuitively subtracts $|L_{q'} \cap L_{q''}|$ from the nominators occurring in the corresponding terms in the equations above which, as such, is not well-defined as this set may be infinite.

It is possible to pinpoint exactly where this approach fails for NFA: the second equality in Eq. 1 simply does not hold for genuinely nondeterministic automata.

### 2.3 A Comparison with Other Metrics

The sum-based distance metric defined in Lem. 1 is based on the sum-based weight function $\omega(L)$ which accumulates the fractions $f_L(n)$ of how many words of length $n$ are included in $L$. These infinitely many fractions are then condensed into a single value by giving non-zero, but exponentially less weight to fractions associated to longer words. This places a lot of emphasis on whether short words are in $L$ or not. Thus, finite languages that contain only short words might end up with a higher weight than infinite languages that contain all but a few short words, depending on the value of $\lambda$.

A different approach to derive one finite value from the infinite sequence $(f_L(n))_{n\in\mathbb{N}}$ is to look at possible limits of this sequence, or of similar sequences. Clearly, already for regular languages the limit $\lim_{n\to\infty} f_L(n)$ need not exist at all (e.g. for the language containing only words of even length). However, this limit,

resp. an asymmetric approximation to it has seen some use in automatic grading of homework assignments [2]. A related version of this approach relates not the number of words in $L^{(n)}$ to $\Sigma^n$, but $\log |L^{(n)}|$ to $n$, where convergence still is not guaranteed. However, in certain settings convergence exists; this goes back to Shannon [14] under the name *channel capacity*, see also [6] for an investigation w.r.t. regular languages. In [8], the upper limit $\limsup_{n\to\infty} \frac{|L^{(n)}|}{n}$ is taken to obtain a size measure for languages, and this is computable for regular languages. Another related notion is that of (conditional) *language density* [12] where the denominator in the fraction $F_L(n)$ can be any regular language.

As mentioned in the introduction, one can derive distance measures on languages from distances on words for instance via $d(L_1, L_2) := \min\{d(w_1, w_2) \mid w_i \in L_i\}$. This results in a distance measure with very different properties compared to the one defined in the previous section which renders them particularly useless for the application presented in the next section. We therefore do not elaborate any further on them. Another possible way is to construct a distance measure on languages based on properties of the shortest word in their symmetric difference. Again, this completely ignores the structure of the difference except for a finite part which we discard here for the same reason.

Finally, one can construct distance measures on languages based on syntactic criteria of their *representations*. Such measures also have their place in e.g. automated grading of homework (cf. [2]) but suffer from their inherent flaw that two representations of the same language may and often will have positive distance. Moreover, they can often be cheated by taking two objects clearly representing different languages, but drowning the difference in useless padding, e.g. by adding similar but unreachable components to two NFAs accepting different languages.

## 3 Similarity in Formal Language Exercises

### 3.1 Automatic Assessment and Grading

Formal language theory is a standard part of the syllabi of computer science programs at universities worldwide. One of the basic competencies taught in corresponding courses is to understand the representation of a possibly infinite language by finite means like automata. A standard exercise on the way to achieving such competencies gives a description of a formal language and asks students to construct an automaton recognising exactly this language. Constructivistic learning theories require the students to be given adequate feedback on their solution attempts in order to initiate error-driven learning cycles that will eventually result in the acquisition of said competencies.

Feedback can be a counterexample to a wrong solution, a hint on where to look for errors, or – in the simplest form – a grade that puts a numerical value onto the assessment of how well the task was solved. Exmploying automatic grading yields several advantages: scalability makes it possible to keep up with growint student numbers and increased demand for exercises, while digitisation yields its typical benefits like increased fairness, faster response times and more focused learning efforts in the absence of human correctors.

Automatic assessment of exercises is an important application for similarity measures on formal languages. As mentioned in the introduction, any distance measure can be turned into a similarity measure by inverting (and possibly scaling) it. In the following, we use *sim* for the similarity measure on languages obtained as $sim(L_1, L_2) = s(1 - d_\lambda(L_1, L_2))$ for the distance measure $d_\lambda$ defined in the previous section and some monotonic function $s$ that maps the interval $[0, 1]$ to a (typically discrete) range of grades or points. We will not discuss the choice of the scaling function $s$, as this is highly dependent on the context dictated by the point scale in use and what teachers may consider to be a "good enough" solution etc. We discuss the choice of a good discounting factor $\lambda$ below.

In comparison to other similarity measures found in the literature and mentioned in the previous section, *sim* features some good properties for the grading task, like well-definedness, effective computability, and – most of all – the fact that it considers two languages to be very similar when they deviate on few words only. There is one distinct disadvantage, though: the exponential decline in the weight function w.r.t. word length puts disproportionately high weights onto short words. Hence, students could achieve high marks by not matching the target language at all, for as long as they cover the shortest words inside and outside of that language. Worst of all, this would give false learning incentives comparable to what can occur in test-based automatic grading, often to be observed for instance in the context of programming exercises [7].

There are two ways to remedy this: one can employ a non-linear scaling function $s$. This can have undesired effects, as it does not distinguish between the two situations in which low grades are achieved either by getting many long words or only some shorts words wrong. We therefore amend *sim* by redistributing weights of short words. This requires students' solutions to capture much larger parts of the target language in order to achieve high numerical similarity values.

### 3.2 Redistribution of Weights on Short Words

While some rebalancing of the individual weights associated to words can be obtained by adjusting the value $\lambda$ that controls how much weight is given to each $\Sigma^n$, this still will assign an exponentially smaller weight to words of length $n + 1$ compared to those of length $n$, since all words of length $n + 1$ together share $\lambda$ times the weight as those of length $n$, but there are $|\Sigma|$ times more of them. Hence, we aim to equalise the weight of all words up to a certain length $\eta$, yielding a weighting scheme defined via

$$\omega_\lambda^\eta(L) = \omega'_\lambda(L^{(\leq\eta)}) + \omega_\lambda(L^{(>\eta)})$$

where $\omega'_\lambda$ satisfies the following.

- $\omega'_\lambda(\{w\}) = \omega'_\lambda(v)$ for all $w, v \in L^{(\leq\eta)}$, i.e. all words of length up to $\eta$ contribute equally to the weight of a language;
- $\omega'_\lambda(\Sigma^*) = \omega_\lambda(\Sigma^*)$, i.e. on the set of all words this really constitutes a redistribution.

For any $L \subsetneq \Sigma^*$, though, we generally have $\omega_\lambda^\eta(L) \neq \omega_\lambda(L)$, and depending on the distribution of words in $L^{(\leq\eta)}$, this rearrangement of weights of words can result in a higher or lower weight of the overall language. We discuss suitable choices for $\eta$ in Sec. 3.4.

In order to compute the rebalancing, we only need the overall weight of all words of length up to $\eta$, and the values of the respective $f_L(n)$ for $0 \leq n \leq \eta$. The first value can be computed straightforwardly as $(1-\lambda)\cdot\sum_{i=0}^{\eta}\lambda^i = 1-\lambda^{\eta+1}$. Since there are $\sum_{i=0}^{\eta}|\Sigma|^i$ many words of length between 0 and $\eta$, the weight of an individual word of such length is $\frac{1-\lambda^{\eta+1}}{\sum_{i=0}^{\eta}|\Sigma|^i}$ under the new weighting scheme. The values of the individual $f_L(n)$ are known to be computable [14,6]; we present here an approach based on the correctness proof of Lem. 2.

**Lemma 3.** *Let $\mathcal{A} = (Q, \Sigma, \delta, q_i, Q_F)$ be a DFA such that $L = L(\mathcal{A})$ and let $n \in \mathbb{N}$. Then the values of $f_L(0), \ldots, f_L(n)$ are computable in combined time $\mathcal{O}(n \cdot |Q|^3 + |Q|^2 \cdot |\Sigma|)$.*

*Proof.* Recall the values $t_{q,q'}$ and $e_q$ from the proof of Lem. 2. Note that the $t_{q,q'}$ define a $|Q| \times |Q|$ matrix $M$. Let $t_{q,q'}^k$ denote the entry in $M^k$ in the row for $q$ and the column for $q'$. Then $t_{q,q'}^1 = t_{q,q'}$. We claim that

$$t_{q,q'}^k = \frac{|\{w \in \Sigma^k \mid \hat{\delta}(q,w) = q'\}|}{|\Sigma|^k} \qquad (\dagger)$$

holds for all $k \in \mathbb{N}$. For $k \leq 1$ this is straightforward. Assume that † holds for $k \geq 1$, we show that it holds for $k+1$. By the definition of matrix multiplication,

$$\begin{aligned} t_{q,q'}^{k+1} &= \sum_{q'' \in Q} t_{q,q''}^k t_{q'',q'} = \frac{|\{w \in \Sigma^k \mid \hat{\delta}(q,w) = q''\}|}{|\Sigma|^k} \cdot \frac{|\{a \in \Sigma \mid \delta(q'',a) = q'\}|}{|\Sigma|} \\ &= \frac{|\{wa \in \Sigma^{k+1} \mid \hat{\delta}(q,wa) = q'\}|}{|\Sigma|^{k+1}} \end{aligned}$$

which proves † for $k+1$. But then $f_L(k)$ is easily computed as $\sum_{q \in Q_F} t_{q_0,q}^k$.

Creating the matrix $M^1$ is done via computing all the $t_{q,q'}$ which takes time in $\mathcal{O}(|Q|^2 \cdot |\Sigma|)$. Computing $f_L(k)$ from $M^k$ takes time in $\mathcal{O}(|Q|)$. The main cost is generated when computing $M^2, \ldots, M^n$. Individual matrix multiplication can be done in time $\mathcal{O}(|Q|^3)$ which yields a time of $\mathcal{O}(n \cdot |Q|^3)$ for all matrices.[1] □

If all the $f_L(k)$ are known for $k \leq \eta$, the rebalanced weight of $L$ is obtained via subtracting the sum of the weights of words under the old weight and re-adding them with their new weight, which yields

$$\omega_\lambda^\eta(L) = \omega_\lambda(L) - \sum_{i=0}^{\eta} f_L(i) \cdot \lambda^i + \sum_{i=0}^{\eta} f_L(i) \cdot c_\lambda^\eta \cdot |\Sigma|^i,$$

where $c_\lambda^\eta = \frac{1-\lambda^{\eta+1}}{\sum_{j=0}^{\eta}|\Sigma|^j}$ is the weight of an individual word of length $\eta$ or less under the new weighting scheme.

[1] Actual matrix multiplication can be done in time $\mathcal{O}(|Q|^{2.37286})$, cf. e.g. [1]. We state the cubic runtime here for the sake of readability.

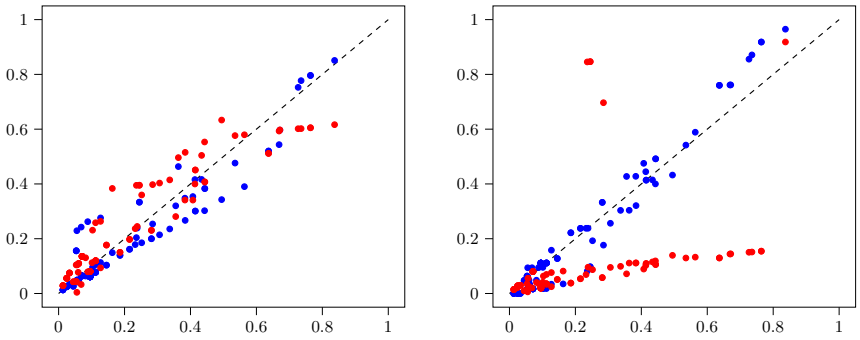


**Fig. 1.** Comparison of the weights of symmetric difference of the submitted automata and $L_{\overline{abba}}$ using $\omega^5_{0.87}$ on the $x$-axis. Left plot: weight using $\omega^0_{0.87}$ (blue), resp. $\omega^{10}_{0.87}$ (red) on the $y$-axis. Right plot: weight using $\omega^5_{0.5}$ (blue), resp. $\omega^5_{0.99}$ (red) on the $y$-axis.

### 3.3 An Implementation and Test Cases

There is an implementation of a procedure that computes similarity values between regular languages based on the weight functions $\omega^\eta_\lambda$, or just the weights of such languages. Inputs are accepted as NFA in a straight-forward format; these are turned into DFA using the standard powerset construction. The implementation is written in Python3 and is publicly available.[2].

Also included are various methods for visualising the effect that different choices of $\eta$ and $\lambda$ have on weight distributions. These can help to determine optimal parameter values in a concrete use case. Finally, we also provide, in this repository, a data set of 754 NFA taken from students' homework exercises asking for the construction of an NFA for some particular language.

On average, similarity of one of these submissions to a model solution was computed in $0.089s$ for a total of 754 submissions on a desktop PC with four 2.40GHz Cores and 8 GB RAM. The tests are run on a single core under Windows 10. Calculations for most automata were made significantly faster thatn $0.089s$ with only a few showing runtimes in the range of seconds.

### 3.4 Empirical Determination of Good Parameter Values

We close this section with a brief discussion on what are good values for the parameters $\eta$ and $\lambda$ in the context of automated grading of homework assignments, using the collected data for the target language $L_{\overline{abba}} := \Sigma^* \setminus \Sigma^* abba \Sigma^*$, where $\Sigma = \{a, b\}$, as a benchmark. Out of the 174 submissions for this exercise, 75 correctly capture $L_{\overline{abba}}$ and are omitted from this discussion, as their distance is 0 under any parameter configuration.

In Fig. 1, we plot the weights of all these automata under different settings for $\eta$ and $\lambda$. The default values are $\lambda = 0.87$ and $\eta = 5$, the latter chosen as the

[2] https://github.com/maurice-herwig/wofa.git

length of the longest acyclic path through the standard 5-state DFA for $L_{\overline{abba}}$. Then $\lambda$ is chosen such that exactly half of the potential weight is assigned to words of length up to $\eta$.

In the left plot of Fig. 1, the distances between a correct DFA for $L_{\overline{abba}}$ and the NFA from the benchmark set are compared to those under two different values of $\eta$, namely 0 (no rebalancing) and 10 (more rebalancing). The $x$-axis denotes weights (of the symmetric difference) under $\omega^5_{0.87}$; the $y$-axis their weights under $\omega^0_{0.87}$ (in blue) resp. $\omega^{10}_{0.87}$ (in red). From the the clustering of points in the left lower corner it is apparent that automata which define a language close to $L_{\overline{abba}}$ receive similar distances in either setting. Some automata with weight around 0.1 under $\omega^5_{0.87}$ have a much higher weight under $\omega^0_{0.87}$, i.e. they benefit from the rebalancing. These are solution attempts that wrongly categorise short words such as $\varepsilon$, which the rebalancing penalises less strongly. For most other automata, rebalancing increases the weight. This is due to mistakes which only manifest themselves on longer words, in particular those that induce cycles in the automata. We conclude from this data set that rebalancing all words up to at least the length of the longest cycle-free path in the automaton is a viable way to make the distance metric put weight onto words more evenly.

For variations of the parameter $\lambda$ (right plot in Fig. 1), we obtain a different picture. Here, the $x$-axis represents automata weights under $\omega^5_{0.87}$, and the $y$-axis shows their weights under $\omega^5_{0.5}$ (in blue) resp. $\omega^5_{0.99}$ (in red). There is little difference between $\lambda = 0.5$ and $\lambda = 0.87$ in the presence of moderate rebalancing. However, the extreme choice of $\lambda = 0.99$ pushes most of the potential weigh out of the rebalancing zone and heavily de-emphasises short words. However, almost all long words do not belong to $L_{\overline{abba}}$ (cf. the limit based distance metrics discussed in Sec. 2.3), whence any automaton that rejects many words due to any reason will receive a low weight under this scheme. The few automata with high weight under this scheme all reject almost no words. Hence, extremely high values of $\lambda$ may not make for a good and levelled similarity measure, especially when many deviations from the target language manifest themselves already on short words, i.e. those that use only one or two cycles in the automaton. All in all, the exact choice of $\lambda$ appears to be less important in the presence of rebalancing.

## 4 Conclusion

We have introduced a new distance metric for formal languages based on convergent geometric series. This guarantees some nice properties, in particular well-definedness and effective computability for regular languages. The techniques employed here are not new; the use of discounting of values of words that decrease exponentially in their lengths can be seen in various concepts found in formal language theory. Yet, the distance metric introduced here, resp. the similarity measures drawn from it, especially after rebalancing the weights of short words, seem to be the first to be used in the application domain of automatic grading, and which provide properties like well-definedness etc.

Work in this area can be continued in several ways. An obvious question that arises asks for an effective way to compute distances between languages represented by nondeterministic models, without explicitly determinising them first. As argued at the end of Sect. 2.2, the method presented here cannot be applied to NFA directly. However, the argument leaves open the possibility that it may work for unambiguous NFA. Besides that, it of course remains to be seen whether the weight of a language of a truly nondeterministic finite automaton can be calculated directly, as this may be useful for efficiency purposes in other application areas. Another question to investigate concerns the nature of the equation systems representing the weights of languages per DFA state. It remains to be seen if these fall into some class for which better solving methods are known, like sparse matrices for instance.

## References


1. Alman, J., Williams, V.V.: A refined laser method and faster matrix multiplication. In: Proc. ACM-SIAM Symp. on Discrete Algorithms, SODA'21. pp. 522–539. SIAM (2021). https://doi.org/10.1137/1.9781611976465.32
2. Alur, R., D'Antoni, L., Gulwani, S., Kini, D., Viswanathan, M.: Automated grading of DFA constructions. In: Proc. 23rd Int. Joint Conf. on A.I., IJCAI'13. pp. 1976–1982. IJCAI/AAAI (2013)
3. Ashby, F.G., Ennis, D.M.: Similarity measures. Scholarpedia **2**(12), 4116 (2007)
4. Choffrut, C., Pighizzini, G.: Distances between languages and reflexivity of relations. Theoretical Computer Science **286**(1), 117–138 (2002). https://doi.org/https://doi.org/10.1016/S0304-3975(01)00238-9, mathematical Foundations of Computer Science
5. Choi, S., Cha, S.H., Tappert, C.: A survey of binary similarity and distance measures. J. Syst. Cybern. Inf. **8** (11 2009)
6. Chomsky, N., Miller, G.A.: Finite state languages. Inf. Control. **1**(2), 91–112 (1958). https://doi.org/10.1016/S0019-9958(58)90082-2
7. Combéfis, S.: Automated code assessment for education: Review, classification and perspectives on techniques and tools. Software **1**(1), 3–30 (2022). https://doi.org/10.3390/software1010002
8. Cui, C., Dang, Z., Fischer, T.R., Ibarra, O.H.: Similarity in languages and programs. Theor. Comput. Sci. **498**, 58–75 (2013). https://doi.org/10.1016/j.tcs.2013.05.040
9. Furht, B. (ed.): Distance and Similarity Measures, pp. 207–208. Springer US (2006). https://doi.org/10.1007/0-387-30038-4_63
10. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, N. Reading, MA (1979)
11. Ifenthaler, D.: Measures of Similarity, pp. 2147–2150. Springer US (2012). https://doi.org/10.1007/978-1-4419-1428-6_503
12. Kozik, J.: Conditional densities of regular languages. Electr. Notes Theor. Comput. Sci. **140**, 67–79 (11 2005). https://doi.org/10.1016/j.entcs.2005.06.023
13. Pearson, W.R.: An introduction to sequence similarity ("homology") searching. Current Protocols in Bioinformatics **42**(1), 3.1.1–3.1.8 (2013). https://doi.org/10.1002/0471250953.bi0301s42
14. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(3), 379–423 (1948). https://doi.org/10.1002/j.1538-7305.1948.tb01338.x