Edit Distance of Finite-Valued Transducers

Prince Mathew Saina Sunny

{prince.mathew, saina.sunny}@ulb.be

Université libre de Bruxelles

ICALP 2026

07 July 2026

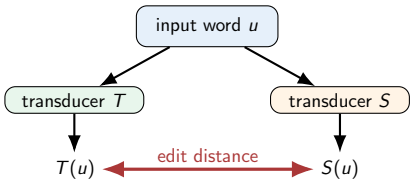
Royal Holloway, University of London

Comparing transformations

Transducers model input-output behaviour

A finite-state transducer reads an input word and produces output word(s):

$$\mathcal{T} \subseteq A^* \times B^*.$$



- Equality asks: are two transformations identical?
- Distance asks: how far apart are they?
- Edit distance is natural for text-producing systems.

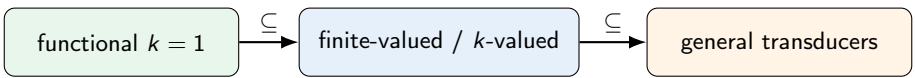
Finite-valued transducers

Definition

A transducer is k-valued if every input has at most k outputs:

$$\forall u \in A^*, \qquad |\mathcal{T}(u)| \leq k.$$

It is finite-valued if it is k-valued for some fixed $k \in \mathbb{N}$.



Why this class?

Many problems that are undecidable in general for transducers become decidable under finite-valuedness.

Problem landscape

Class of transducers	Equivalence / inclusion	Edit distance
Functional	decidable [Gurari et. al. 1983]	computable [Aiswarya et. al. 2024]
Finite-valued	decidable [Culík II et al. 1986]	?
General transducers	undecidable [Fischer et al. 1968]	uncomputable [Aiswarya et. al. 2024]

Question

Is there a more expressive and well-behaved class beyond functional transducers where edit distance remains computable?

Problem landscape

Class of transducers	Equivalence / inclusion	Edit distance
Functional	decidable [Gurari et. al. 1983]	computable [Aiswarya et. al. 2024]
Finite-valued	decidable [Culík II et al. 1986]	?
General transducers	undecidable [Fischer et al. 1968]	uncomputable [Aiswarya et. al. 2024]

Question

Is there a more expressive and well-behaved class beyond functional transducers where edit distance remains computable?

Yes - finite-valued transducers.

Distance between transducers

For a word metric d and relations $R, S \subseteq A^* \times B^*$:

$$\mathrm{d}(R,S) = \begin{cases} \sup_{u \in \mathrm{dom}(R)} H_{\mathrm{d}}\big(R(u), S(u)\big), & \mathrm{dom}(R) = \mathrm{dom}(S), \\ \infty, & \text{otherwise.} \end{cases}$$

Here, H_{d} is the Hausdorff distance, that lifts distance between words to sets of words.

Intuition

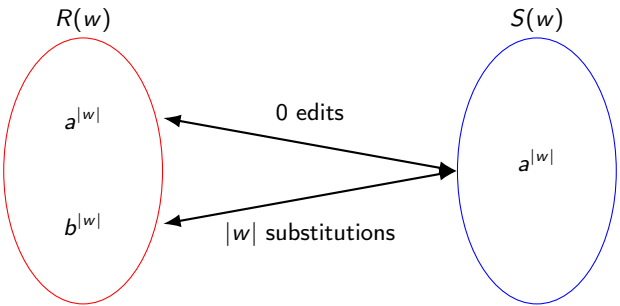
The distance between R and S is the minimal integer K such that, for every input u,

- every output of $R(u)$ is within K edits[1] of some output of $S(u)$, and
- every output of $S(u)$ is within K edits of some output of $R(u)$.

[1] edit operations include insertion, deletion, letter-to-letter substitution, $\cdots$

Distance Between Relations

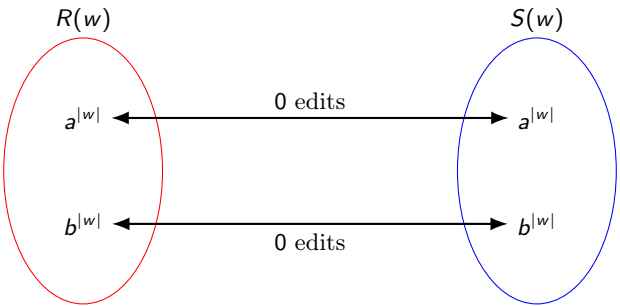
Let $R(w) = \{a^{|w|}, b^{|w|}\}$ and $S(w) = \{a^{|w|}\}$.



$$\mathrm{d}(R, S) = \infty.$$

Distance Between Relations

Let $R(w) = \{a^{|w|}, b^{|w|}\}$ and $S(w) = \{a^{|w|}, b^{|w|}\}$.

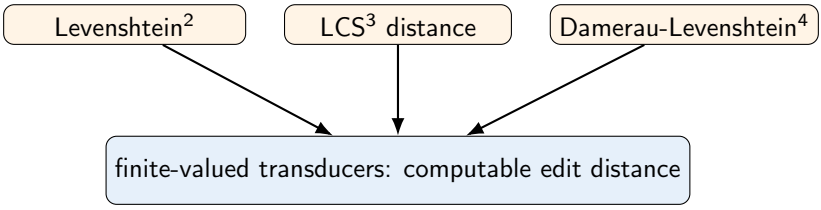


$$\mathrm{d}(R, S) = 0.$$

Main theorem

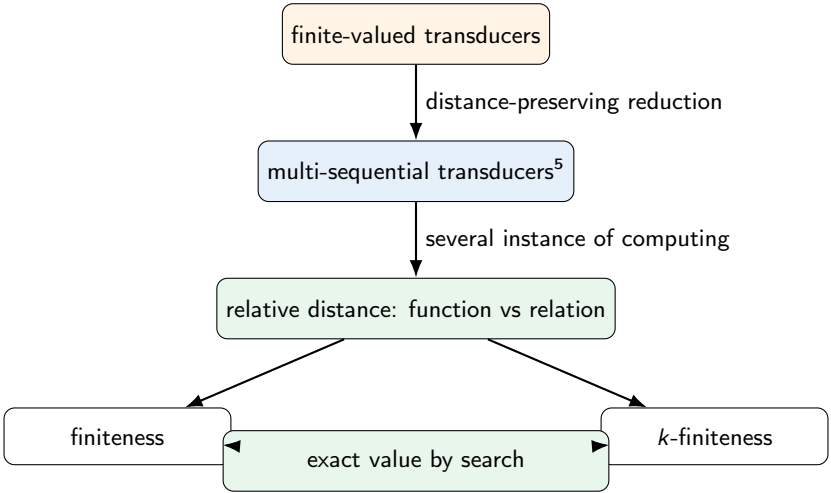
Theorem

Let T and S be finite-valued transducers. For every distance in the Levenshtein family, $\mathrm{d}(T, S)$ is computable.



[2] Levenshtein — insertion, deletion and substitution
[3] LCS — insertion and deletion
[4] Damerau Levenshtein — insertion, deletion, substitution and swapping adjacent letters

High-level proof strategy



[5] finite union of sequential transuducers (those that are detereministic in the input)

Step 1: finite-valued to multi-sequential

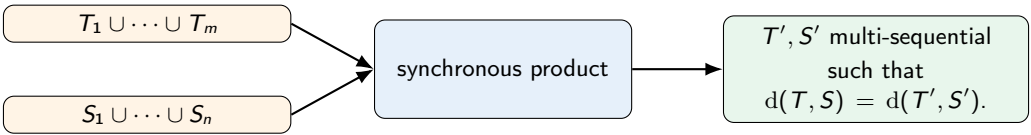
Every finite-valued transducer can be decomposed into a finite union of functional transducers:

$$T \equiv T_1 \cup \cdots \cup T_m, \qquad S \equiv S_1 \cup \cdots \cup S_n, \qquad \mathrm{dom}(T) = \mathrm{dom}(S)$$

The construction builds the multi-sequential transducers T' and S' with a common underlying product automaton $\mathcal{A}$ such that

- $\mathcal{A}$ synchronizes runs of all components of T and S.
- Its input alphabet are tuples of component transitions, making $\mathcal{A}$ deterministic.
- Its states records the state reached by the current run and the set of reachable states in each of the components.
- Its accepting states ensure that all accepting component runs are represented by a global run of $\mathcal{A}$.

$$\{(T(w), S(w)) \mid w \in \mathrm{dom}(T)\} = \{(T'(w), S'(w)) \mid w \in \mathrm{dom}(T')\}$$



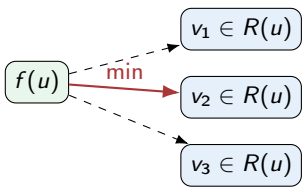
Step 2: multi-sequential distance via relative distance

Definition (Relative distance)

For a function f and a relation R,

$$\overrightarrow{\mathrm{d}}(f, R) = \begin{cases} \sup\limits_{u \in \mathrm{dom}(f)} \inf\limits_{v \in R(u)} \mathrm{d}\big(f(u), v\big), & \mathrm{dom}(f) \subseteq \mathrm{dom}(R), \\ \infty, & \text{otherwise.} \end{cases}$$

- Standard distance between a function and a relation is too strong.
- It requires comparing the function output to *all* outputs of the relation.
- Relative distance asks for the closest output of the relation.



choose the closest output on each input

Step 2: multi-sequential distance via relative distance

Let $R = f_1 \cup \cdots \cup f_m$, $S = g_1 \cup \cdots \cup g_n$, where each f_i, g_j is sequential.

Definition (Relative distance)

For a function f and a relation R,

$$\overrightarrow{\mathrm{d}}(f, R) = \begin{cases} \sup\limits_{u \in \mathrm{dom}(f)} \inf\limits_{v \in R(u)} \mathrm{d}\big(f(u), v\big), & \mathrm{dom}(f) \subseteq \mathrm{dom}(R), \\ \infty, & \text{otherwise.} \end{cases}$$

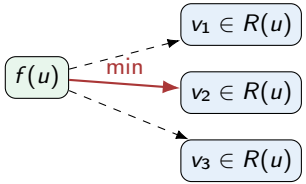
Reduction

If $\mathrm{dom}(R) = \mathrm{dom}(S)$, then

$$\mathrm{d}(R, S) = \max\Big(\{\overrightarrow{\mathrm{d}}(f_i, S) : i \in [m]\} \cup \{\overrightarrow{\mathrm{d}}(g_j, R) : j \in [n]\}\Big).$$

Why relative distance is the right notion

- Standard distance between a function and a relation is too strong.
- It requires comparing the function output to *all* outputs of the relation.
- Relative distance asks for the closest output of the relation.



choose the closest output on each input

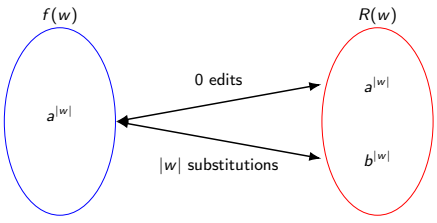
Example

Let $R = f \cup g$, where

$$f(w) = a^{|w|}, \qquad g(w) = b^{|w|}.$$

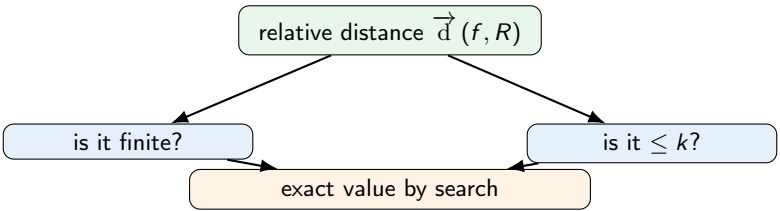
Then $\mathrm{d}(R, R) = 0$, but $\mathrm{d}(f, R) = \infty$ under the usual edit distance. Relative distance fixes this:

$$\overrightarrow{\mathrm{d}}(f, R) = 0.$$



Step 3: Computing relative distance

The exact value of $\overrightarrow{\mathrm{d}}(f, R)$ is computed by reducing to two decision problems.



Key point

For a sequential function f and a multi-sequential relation R, both decision problems are decidable for the Levenshtein family of distances.

Finiteness: structural characterisation

The finiteness test uses the known connection between bounded edit distance between sequential transducers and conjugacy of loop outputs.

Definition

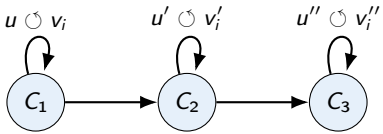
Words u and v are conjugate if $u = xy$ and $v = yx$ for some words x, y.

Theorem (Aiswarya et al. 2024)

Let T and S be functional transducers. For every distance in the Levenshtein family, $d(T, S) < \infty$ iff $\mathrm{dom}(T) = \mathrm{dom}(S)$ and every pair of output words generated by loops in the (trim) Cartesian product of T and S is conjugate.

Finiteness: structural characterisation

Let f be a sequential function and $R = g_1 \cup \cdots \cup g_n$, where each g_i is sequential.

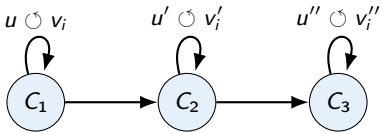


one common index i witnesses boundedness along the path

For the product of f with the components of R

Finiteness: structural characterisation

Let f be a sequential function and $R = g_1 \cup \cdots \cup g_n$, where each g_i is sequential.



one common index i witnesses boundedness along the path

For the product of f with the components of R :

- decompose the product into directed acyclic graph of SCC paths.
- along each path, find one component of R that stays boundedly close to f.
- boundedness follows from conjugacy of corresponding loop outputs.

k-finiteness: finite-state search over edit budgets

To check whether $\overrightarrow{\mathrm{d}}(f, R) \leq k$, build an NFA $A_{C,k}$.

Idea

$A_{C,k}$ reads an input word and simulates the product of f with the components of R. It stores:

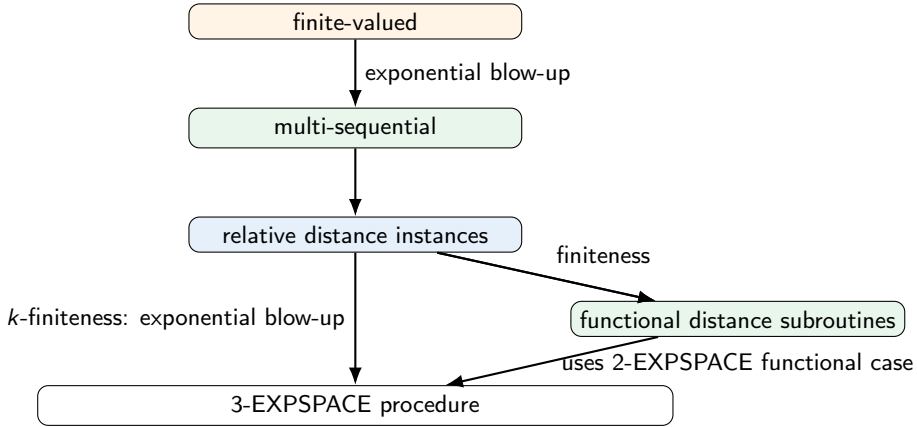
- a remaining edit budget for each component;
- the unmatched leftover between the output of f and that component;
- a word is accepted if it reaches a state where at least one component has a non-negative remaining budget.

Then:

$$\overrightarrow{\mathrm{d}}(f, R) \leq k \qquad \Longleftrightarrow \qquad L(A_{C,k}) = \mathrm{dom}(f).$$

So the problem reduces to language equivalence for finite automata.

Complexity



Lower bound

Computing edit distance is at least PSPACE-hard, since equivalence reduces to checking whether the distance is zero.

Conclusion

Main contribution

We show that the edit distance of finite-valued transducers is computable for the Levenshtein family of distances.

Future work

- Optimise the efficiency of the construction.
- Extend the approach to other distances: Hamming distance, transpositions, and beyond.
- Study edit distance of broader classes of rational and regular relations.

Thank you!