# Large-scale weighted sequence alignment for the study of intertextuality in Finnic oral folk poetry

**Maciej Janicki**

University of Helsinki, Finland

Corresponding author: Maciej Janicki , maciej.janicki@helsinki.fi

**Abstract**

The digitization of large archival collections of oral folk poetry in Finland and Estonia has opened possibilities for large-scale quantitative studies of intertextuality. As an initial methodological step in this direction, I present a method for pairwise line-by-line comparison of poems using the weighted sequence alignment algorithm (a.k.a. 'weighted edit distance'). The main contribution of the paper is a novel description of the algorithm in terms of matrix operations, which allows for much faster alignment of a poem against the entire corpus by utilizing modern numeric libraries and GPU capabilities. This way we are able to compute pairwise alignment scores between all pairs from among a corpus of over 280,000 poems. The resulting table of over 40 million pairwise poem similarities can be used in various ways to study the oral tradition. Some starting points for such research are sketched in the latter part of the article.

**Keywords**

algorithm optimization; alignment; edit distance; oral tradition; poetry; text similarity

## I INTRODUCTION

The digitization of vast collections of Finnic Kalevalametric oral folk poetry has opened new possibilities for large-scale studies utilizing computational and quantitative methods. The collections *Suomen Kansan Vanhat Runot*[1] (*Old Poems of the Finnish People*) and *Eesti Regilaulude Andmebaas*[2] (*Estonian Runosongs' Database*) contain almost 100,000 items (texts) each. Recently those two national collections have been combined into a single corpus, as well as extended with further archival materials and literary works, to a collection of over 280,000 items within the Digital Humanities project FILTER[3] [Kallio et al., 2023].

Texts collected from oral tradition are usually recordings of a performance. Motifs and storylines circulate within the tradition and are put together by a singer to a unique combination [Honko, 2000] – even two performances of the same epic cycle by the same singer may differ considerably. The process of oral transmission, performance and collection leads to a text corpus containing a plenty of partial similarities, but often in an obfuscated form, which is not trivial to detect automatically.

In this article, I consider the problem of automatic detection of textual similarities in the above-mentioned corpus. This is the first step towards a large-scale quantitative study of intertextuality,

[1] https://skvr.fi

[2] https://www.folklore.ee/regilaul/andmebaas/

[3] Formulaic Intertextuality, Thematic Networks and Poetic Variation across Regional Cultures of Finnic Oral Poetry. Academy of Finland grant No. 333138.

which could give new insights into the Finnic oral tradition. Building on earlier work on measuring line similarity and detecting equivalent lines [Janicki et al., 2023], here I apply the weighted sequence alignment algorithm to align texts line-by-line and measure their similarity based on the alignment. The main challenge is to optimize the algorithm so that computing alignments of all text pairs within a large collection becomes feasible.

The present article is an extended version of a short conference paper [Janicki, 2022]. While the former focused on general and technical description of the algorithm optimization, here I provide more information on the context in which it is applied. However, the optimization of the alignment algorithm remains the main contribution of this paper as well. An implementation of the resulting algorithm is publicly available.[4]

## II RELATED WORK

**Weighted Sequence Alignment.** The Weighted Sequence Alignment algorithm was presented independently by at least Needleman and Wunsch [1970] and Wagner and Fischer [1974] and is based on the concept of 'edit distance' [Levenshtein, 1966]. As the algorithm is quite simple, numerous libraries contain an implementation of it. However, most available Python packages for sequence alignment are either designed specifically for biological sequences (like e.g. `Bio.Align`[5]) or very simple (and thus inefficient) pure-Python implementations (like e.g. `alignment`[6], `edit-distance`[7]). A notable example of a library allowing for alignment of sequences of numeric vectors using a custom similarity measure, as well as providing a fast C++ implementation, is `pyalign`[8]. However, as the benchmarks in Janicki [2022] have shown, it does not provide sufficient performance to solve the problems addressed here.

Optimizations to the base algorithm are typically based on restricting the allowed edit distance to a small number and pre-selecting or filtering candidate pairs [e.g. Bocek et al., 2007, Soru and Ngonga Ngomo, 2013]. For handling large numbers of strings, also finite state automata have been used Schulz and Mihov [2002]. However, these methods are only applicable to sequences of symbols from a finite alphabet.

**Text alignment and reuse.** The concept of text alignment for the purpose of comparison of similar texts has been present in Digital Humanities for a long time. Examples of it include the Versioning Machine [Schreibman et al., 2003], the Tesserae project for aligning Latin poetry [Coffee et al., 2013], innovative visualizations for medieval French poetry [Jänicke and Wrisley, 2017], or recently the Reception Reader for browsing text reuse in a corpus of early modern English publications [Rosson et al., 2023]. The quantitative study of intertextuality is an emerging research subject that builds upon automatic discovery of similarities in large collections of text or other media (see Forstall and Scheirer [2019] for a general introduction).

## III OPTIMIZING THE WEIGHTED SEQUENCE ALIGNMENT ALGORITHM

### 3.1 The base algorithm

The starting point for our procedure is the weighted edit distance algorithm described by Wagner and Fischer [1974]. However, while the base algorithm is formulated in terms of 'distance' (thus

[4]https://github.com/maciejjan/matrix-align
[5]https://biopython.org/docs/1.75/api/Bio.Align.html
[6]https://pypi.org/project/alignment/
[7]https://pypi.org/project/edit-distance/
[8]https://pypi.org/project/pyalign/

0 meaning complete similarity and 1 complete dissimilarity of individual units), the formulation applied here interprets the weight as ‘similarity’ (thus 1 meaning complete similarity (identity) and 0 complete dissimilarity). This means that also the weight of insertions and deletions is 0, which is a necessary assumption for the following optimization. In this formulation, we are looking for the maximum-weight alignment, which detects as much overlap between the two sequences as possible.

Let $n_1, n_2$ denote the length of the sequences to be aligned, and $S$ denote the $n_1 \times n_2$ matrix of similarities[9] between individual units of both sequences. We are going to compute another $n_1 \times n_2$ matrix $D$, with $d_{i,j}$ being the optimal alignment weight of the first $i$ elements of the first sequence and first $j$ elements of the second sequence. The alignment matrix $D$ can be computed using the following recursive formula [cf. Wagner and Fischer, 1974]:

$$d_{i,j} = \max \left\{ \begin{array}{c} d_{i-1,j} \\ d_{i,j-1} \\ d_{i-1,j-1} + s_{i,j} \end{array} \right\} \tag{2}$$

where the considered values amount to the edit operations of deletion, insertion and substitution, respectively. After computing the matrix $D$, the weight of the optimal alignment can be found in its bottom-right corner, while the alignment itself can be retrieved by backtracing, from which direction the maximum value was chosen at each step.

In the subsequent presentation we are going to adapt the terminology to the use case described in this article, thus speaking of ‘poems’ instead of ‘sequences’ and ‘verses’ instead of ‘units’. However, the same sequence alignment algorithm could be applied in different fields (e.g. bioinformatics), where the sequences and units could be defined differently.

### 3.2 Optimization

Our optimization is based on the idea that computation on vectors and matrices is faster than computing individual numbers iteratively, especially when using a GPU. We will thus group the computations in two ways:

1. Use vector operations to compute entire rows of the alignment matrix.
2. Use matrix operations to compute the next row of alignment matrices between one poem and all other poems at once.

**Optimization 1.** Because in the formula (2) every cell of the matrix $D$ depends on the cell to the left, we cannot use it directly to compute entire rows. However, we can break down this computation into two stages:

$$d^*_{i,j} = \max \{ d_{i-1,j} \; ; \; d_{i-1,j-1} + s_{i,j} \} \tag{3}$$

$$d_{i,j} = \max \left\{ d^*_{i,j} \; ; \; d_{i,j-1} \right\} = \max_{k \le j} d^*_{i,k} \tag{4}$$

[9]In our use case, we use the cosine similarity of character bigrams as a similarity measure of lines, following Janicki et al. [2023]. However, the alignment algorithm only requires a measure that produces a positive number, with 0 meaning ‘no similarity’.

Because cosine similarity below 0.5 is easily achieved by unrelated lines and thus such numbers do not correspond to any real similarity, we apply an additional *rescaling* step as follows:

$$S_{\text{rescaled}} = \max \left\{ 0, \frac{S_{\text{cos}} - 0.5}{0.5} \right\} \tag{1}$$

with $S_{\text{cos}}$ being the matrix of original cosine similarities, and $S_{\text{rescaled}}$ the similarity matrix that we use in the algorithm.

Now (3) depends only on the previous row, so it can be computed row-wise, whereas (4) is a cumulative maximum operation. Let fmax$(\cdot;\cdot)$ denote the element-wise maximum of two vectors or matrices and cummax$(\cdot)$ the cumulative maximum (row-wise in case of matrices). Then we can rewrite (3, 4) in vector notation as:

$$d^*_{i,1:n} = \text{fmax}\begin{pmatrix} d_{(i-1),1:n} \\ d_{(i-1),0:(n-1)} + s_{i,1:n} \end{pmatrix} \tag{5}$$

$$d_{i,0:n} = \text{cummax}\left(d^*_{i,0:n}\right) \tag{6}$$

Note that the latter step (cummax) relies on the fact that the insertion weight is 0, and the optimization could not be applied otherwise.

**Optimization 2.** Assuming that we are computing the alignment between a single *target* poem and multiple *source* poems, the next row for each source poem can be computed at once. We will stack the matrices $S$ and $D$ vertically, so that the columns correspond to the verses of the target poem and the rows to the verses of all source poems concatenated.[10] Let $B$ denote a set of sequence boundaries, i.e. row indices in the stacked matrices, at which a new poem begins. Further, let $m, n$ denote the (zero-based) indices of the last row and column of the $D$ and $S$ matrices.

**Algorithm 1** Alignment of a single poem against multiple others.

```
1: $D_{B,0:n} \leftarrow \text{cummax}(S_{B,0:n})$
2: $I \leftarrow (B+1) \setminus B$
3: while $I \neq \emptyset$ do
4:     $D_{I,0} \leftarrow \text{fmax}\left(D_{I-1,0}, S_{I,0}\right)$
5:     $D_{I,1:n} \leftarrow \text{fmax}\begin{pmatrix} D_{I-1,1:n} \\ D_{I-1,0:n-1} + S_{I,1:n} \end{pmatrix}$
6:     $D_{I,0:n} \leftarrow \text{cummax}(D_{I,0:n})$
7:     $I \leftarrow (I+1) \setminus B \setminus \{m+1\}$
8: end while
```

Algorithm 1 computes the stacked alignment matrix $D$. Each iteration computes the next row of the alignment matrix for each source poem simultaneously. The set $I$ contains the indices of currently computed rows. The notation like $I + 1$ for a set of indices is a shorthand for $\{i + 1 : i \in I\}$. Once an index reaches the start of a new poem or the end of the corpus, it is removed from the set (line 7). The first row for each poem (line 1) and the first column (line 4) are processed separately as they cannot refer to the previous row or column, respectively.

The detailed benchmarks demonstrating the effect of the optimization are provided in [Janicki, 2022, sec. 3]. The version using both above optimizations on GPU is at least 13x faster than pairwise comparison using `pyalign` and at least 50x faster than the naïve Python implementation.[11]

[10]Because the alignment is symmetric, assuming that the goal is to compute alignment between all poem pairs and thus we will take every poem in turn to be a target poem, it suffices if the source poems all *follow* the target poem in the corpus (rather than the entire rest of the corpus).

[11]The reported factors are for dataset sizes of 10,000 and 2,000 poems, respectively. The speedup factor grows with the size of the dataset and for larger sizes the less efficient computations could not be performed in reasonable time.

### 3.3 Similarity measures

After running Algorithm 1, we can extract from the last column of the matrix $D$ the vector $D_{Q,n}$, with $Q = (B \setminus \{0\} \cup \{m+1\}) - 1$ being the set of row indices corresponding to the last verse of each poem. Then, $D_{Q,n}$ represents the weight of the optimal alignment (sum of similarities of all aligned verse pairs) between the target poem and every other poem in the corpus. We will call it the *raw similarity*.

For given two poems, let $s_{\text{raw}}$ denote their raw similarity. This figure tells us roughly how many aligned verses do the two poems have in common (with non-identical verse pairs counting as less than one, adequately to their similarity). However, it is not very informative as a similarity measure because we don't know how large part of the poems the aligned verses represent. Thus, let $n_1, n_2$ denote the lengths of the two poems. The *one-sided similarities*:

$$s_1 = \frac{s_{\text{raw}}}{n_1} \tag{7}$$

$$s_2 = \frac{s_{\text{raw}}}{n_2} \tag{8}$$

tell us, how large part of either of the poems is contained in the other one.

Finally, in order to obtain a symmetric similarity measure (denoted simply by $s$), we calculate the harmonic mean of both one-sided similarities, which tells us how large part of both poems is aligned if they are displayed side-by-side:

$$s = \frac{2}{\frac{1}{s_1} + \frac{1}{s_2}} = \frac{2s_{\text{raw}}}{n_1 + n_2} \tag{9}$$

### 3.4 Alignment extraction

For some applications, it might be necessary to store not just the poem-level similarity figures, but also the verse-by-verse alignment. Similarly to the base algorithm, the optimal alignment can be easily retrieved from the matrix $D$ and also this step can be expressed as matrix operations.

In the original Wagner-Fischer algorithm the alignment is extracted by starting in the bottom-right corner of the alignment matrix $D$ and moving at every step in the direction, from which the maximum in formula (2) was taken (top, left or top-left), until the top-left corner of the matrix is reached. In the optimized version we are going to apply the same approach, but simultaneously for each poem represented in the matrix $D$.

The procedure is illustrated in Algorithm 2. It returns two vectors: $a_{1:m}$ contains for each of the $m$ lines in the corpus the number of the line from the target poem that it is aligned to (i.e. a number between $1$ and $n$), or $-1$ if it is not aligned to any. The corresponding vector $w_{1:m}$ returns the weights (similarities) of the aligned pairs, or $0$ for unaligned lines. Thus the vectors $a_{1:m}$ and $w_{1:m}$ together represent the alignment between the target poem and all other poems in the corpus.

We see the alignment matrix $D$ as a vertical concatenation of component matrices corresponding to poem pairs, with $B$ pointing to the row indices where a new component starts. When extracting the alignments, we traverse all components simultaneously. Thus, the vectors $i$ and $j$ contain the current row and column indices in all component matrices. They are initialized to the bottom-right corners of the components, i.e. row indices to the indices preceding the start of a new component (line 3) and column indices to the rightmost column (line 4).

**Algorithm 2** Extraction of the alignment from the matrix $D$.

```
1:  $a_{0:m} \leftarrow -1$
2:  $w_{0:m} \leftarrow 0$
3:  $i_{0:m} \leftarrow (B-1) \setminus \{0\} \cup \{m\}$
4:  $j_{0:m} \leftarrow n$
5:  while length($i$) $> 0$ do
6:      $u \leftarrow (i \notin B)$
7:      $v \leftarrow (j > 0)$
8:      $q \leftarrow (D_{i,j} = D_{i-1,j}) * u$
9:      $r \leftarrow (D_{i,j} = D_{i,j-1}) * v * (1-q)$
10:     $s \leftarrow (1-q) * (1-r)$
11:     $a_{i_s} \leftarrow j_s$
12:     $w_{i_s} \leftarrow d_{i_s,j_s} - d_{i_s-1,j_s-1} * u_s * v_s$
13:     $i \leftarrow i - q - s$
14:     $j \leftarrow j - r - s$
15:     $k \leftarrow (i \notin (B-1) \setminus \{0\}) * (i \geq 0) * (j \geq 0)$
16:     $i \leftarrow i_k$
17:     $j \leftarrow j_k$
18: end while
```

In a single iteration of the loop, we determine whether to shift the indices to the left, top or top-left. For that we introduce the following auxiliary variables: $u$ checks whether the current position is *not* in the topmost row of the component, and $v$ whether it is *not* in the leftmost column. Both are binary vectors of the same size as $i$ and $j$, containing a $1$ at positions where the condition is fulfilled and $0$ elsewhere. Using those, we can calculate further binary vectors: $q$ indicates positions in which the optimal value came from the *top*, $r$ those where it came from the *left* and $s$ those where it came from the *top-left*. Note that at each position, exactly one of $q, r, s$ has to be 1. If the value at the current position is equal to the one above it, and the latter is in the same component matrix (i.e. the current row is not the topmost), then the optimal value came from above (line 8). Otherwise, if the current value is equal to the one of the left (provided that there is a column to the left), then it came from the left (line 9). Finally, in all other cases it has to come from the top-left (line 10).

The indices of aligned verse pairs are given by $i_s, j_s$.[12] We use them to set the vectors $a$ and $w$ at the currently considered positions (lines 11-12), with '$*$' denoting element-wise vector multiplication. Then we shift the indices according to the direction from which the optimal values came (lines 13-14). Finally, we remove from the vectors $i, j$ the indices that have crossed the boundary of their component (lines 15-17). We repeat the procedure until $i, j$ are empty, which means that all components have been traversed.

## IV QUANTITATIVE OVERVIEW

The algorithm above has been run on our corpus of 284,331 poems, totalling 4,557,870 verses. I have restricted the size of the results by applying thresholds: for each pair of poems, the criteria $s_{\text{raw}} \geq 1$ and $\max\{s_1, s_2\} \geq 0.1$ had to be fulfilled (see Sec. 3.3). As we haven't designed a formal evaluation method yet, those criteria were intended to be loose and prioritize completeness over correctness, as tighter filtering criteria can always be applied afterwards.

[12]For a vector $i$ and a binary vector $s$ of the same size, $i_s$ means the elements of $i$ at positions, at which $s = 1$.

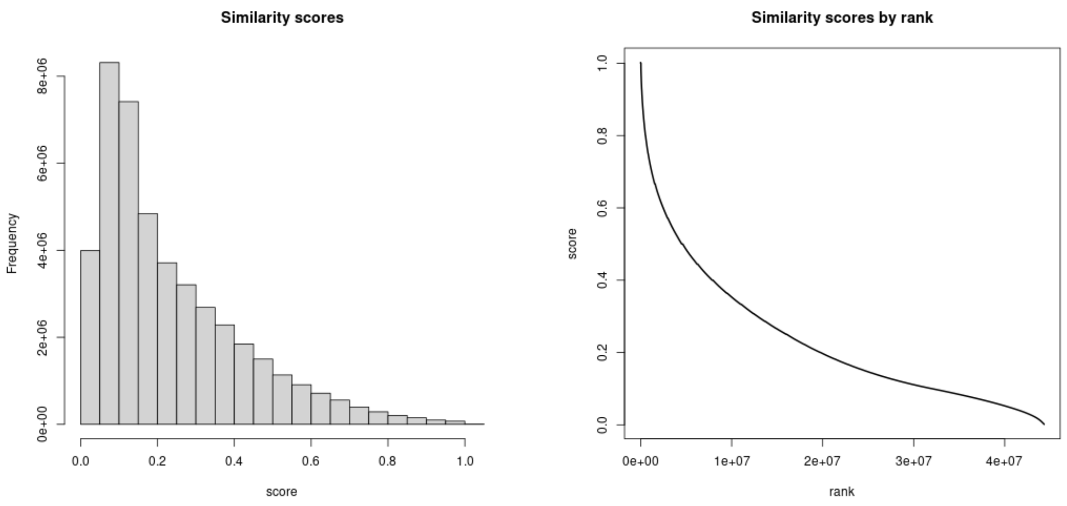


Figure 1: The distribution of symmetric similarity scores (left: histogram, right: rank-value plot).

The computation was split into 2 parallel processes and took around 70 hours on a GPU-equipped computing cluster. The result were two tables: a table of poem similarities containing 41,059,771 poem pairs and a table of verse-level alignments containing 179,735,571 aligned verse pairs. A detailed quantitative analysis of those results is expected to give new insights into the Finnic oral tradition: the spread of motifs, poetic formulas and larger text passages across time and place. Below I present a short introductory overview of the results, while in Sec. V I sketch some ideas for a more detailed analysis, which will be developed in the following publications.

Figure 1 shows the overall distribution of symmetric similarity scores. The peak appears to be around 0.1, which is probably the product of the filtering criteria ($\max\{s_1, s_2\} \geq 0.1$). However, all similarity ranges include large numbers of pairs – e.g. from the right-hand plot it can be read that there is almost 10 million pairs with similarity above 0.4.[13] Figure 2 shows the number of neighbors (entries in the similarity table) per poem as a rank plot. The vast majority of poems has between 10 and 1000 neighbors. Finally, Figure 3 shows the distribution of similarity scores of a poem to its closest neighbor. It is much more even than the one shown in Fig. 1, but it also appears bimodal, with the mode around 1 corresponding to poems that have an exact duplicate (with possible slight spelling differences) and the one around 0.2 to similarities typical for oral transmission. The overall distributions show that textual similarity is abundantly present in the oral folk poetry collections.

In addition to poem texts the collections contain metadata, like the place in which the poem was collected (parish and county), name of the collector and year of the collection. Further, the SKVR and ERAB corpora contain manually constructed type indices, which classify poems as containing one or more *types*, i.e. recurring semantic entities (e.g. an epic story, a lyrical motif, a song for a certain occasion, a charm for a certain purpose).[14] By comparing the computed textual similarity with the metadata fields, we can get an overall picture of the relationships between the texts recognized as similar.

In Figure 4 the pairs of similar poems were divided into groups, based on the symmetric similarity

[13]The rank plot is made from the list of values sorted in decreasing order, by plotting the position in the list (rank) against the value.

[14]While the SKVR type index is considered complete, the ERAB index is work in progress and might be missing annotations for a significant number of poems.

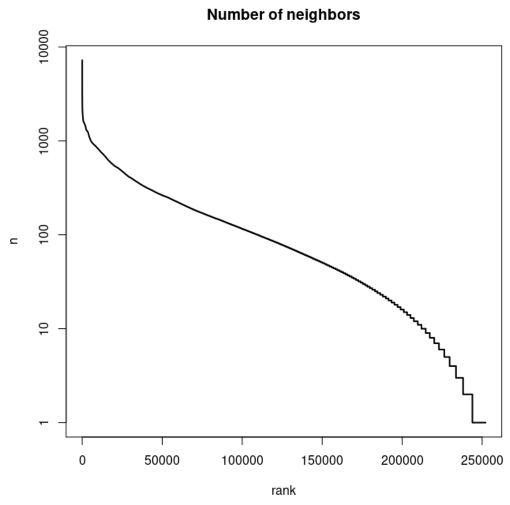


Figure 2: Number of neighbors per poem (rank-value plot; poems with no neighbors excluded).

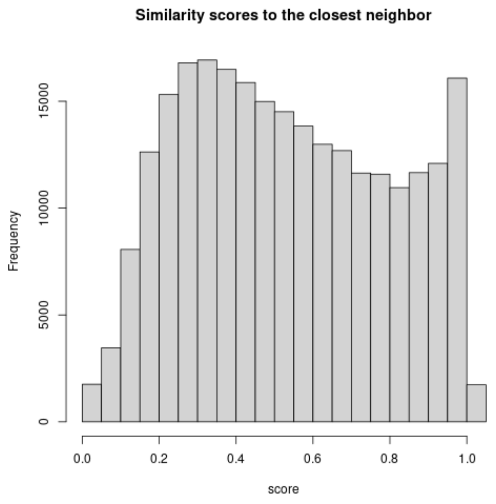


Figure 3: The distribution of similarity scores to the closest neighbor, by poem.

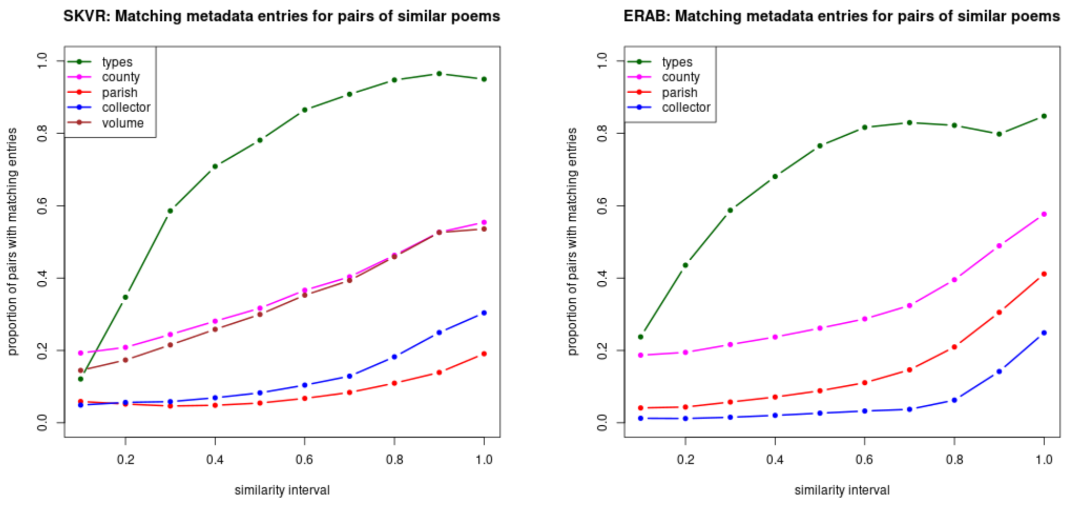


Figure 4: Matching metadata entries for pairs of similar poems (left: SKVR, right: ERAB).

score rounded to one significant digit (e.g. all pairs with similarity between 0.2 and 0.3 were assigned to the group ‘0.2’). For each group we measure the proportion of pairs with equal values in the various metadata fields (county, parish, type, collector, volume[15]). It is generally expected that the curves go up, i.e. with increasing textual similarity there is an increasing chance that the poems were collected in the same place by the same collector and are similar in terms of content (types). However, looking at the exact values, one can see that the similarity is by far not limited to poem pairs coming from the same parish, or annotated with the same types.

## V POSSIBLE APPLICATIONS

The poem similarity table can be used for studying oral poetry from different angles. While on one hand it is a large quantitative dataset enabling global views on the textual similarity within entire collections, on the other hand the numbers are interpretable, i.e. they can be traced to individual pairs of poems, the similarity of which can be inspected by close-reading.

Below I list several subjects currently being studied in the FILTER project, in which the automatically computed similarity is highly useful. As each of them requires combining a fine-grained quantitative analysis with qualitative research on the folklorist side, they are presented here in a very abbreviated description, while they will be developed in further articles.

**Interactive browsing.** For the interactive exploration of the results, we have designed a Web user interface called Runoregi[16] (introduced by Janicki et al. 2023), which has been used and developed constantly for the last 3 years. The algorithm in the present paper contributes a precomputed table of poem similarities, which is displayed in Runoregi as a ‘Similar poems’ box (Fig. 5, bottom-left), the percentage scores being the symmetric similarity (9). Clicking on one of the poems in the list leads to an alignment view (Fig. 5, right).

The similarity scores can also be viewed interactively in other ways. For example to see relationships within a larger set of poems simultaneously – e.g. all poems sharing a certain type annotation – we can apply hierarchical clustering based on the similarity scores (Fig. 6, left). The inner nodes of the resulting dendogram are enriched with links (blue squares) leading to a view that shows the multiple sequence alignment of all poems in a given subtree (Fig. 6, right). Another option is a network view showing a breadth-first search starting from a certain poem (Fig. 7).

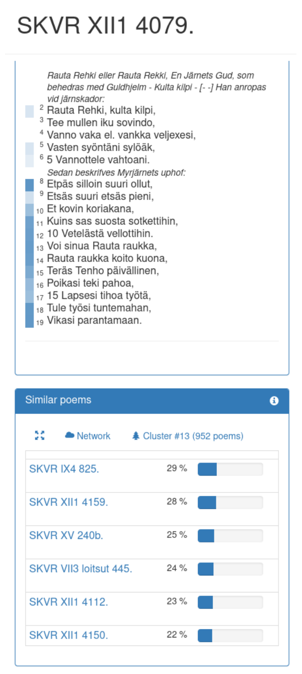
**Completing the typologies.** The similarity table can be used to propose type annotations for poems that have none. As previously mentioned, the type index of the Estonian Runosongs’ Database is work in progress and currently there are 37,560 poems (34%) with no type annotation. Although the SKVR index is considered completed, still 1,263 poems without annotation (1.4%) were found. Finally, the part of the corpus containing unpublished Finnish archival materials consists of 85,248 poems with no indexing, but most of them bearing similarity to some poem from SKVR.

The work on type indices can be facilitated by proposing for each unannotated poem the types of its closest annotated neighbor (often a near-duplicate). Such a list still needs to be reviewed manually. This workflow is currently being tested on the small unannotated part of SKVR.

**Oral-literary relationships.** In addition to materials collected directly from oral tradition, our corpus contains also published works – from small compilations like D. E. D. Europaeus’

[15]The ‘volume’ is given for SKVR, which has been published as a printed collection. The book volumes are arranged by county, which is why the curve is very similar to the one for county.

[16]https://runoregi.rahtiapp.fi



SKVR XII1 4079.

*Rauta Rehki eller Rauta Rekki, En Järnets Gud, som behedras med Guldhjelm - Kulta kilpi - [- -] Han anropas vid järnskador:*

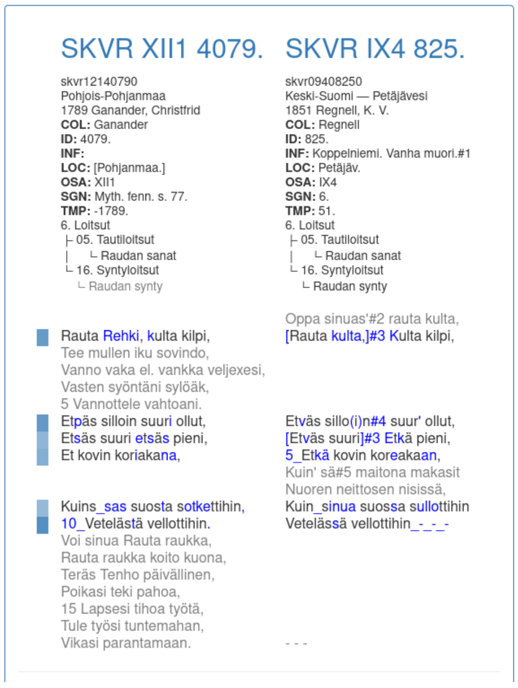
2 Rauta Rehki, kulta kilpi,
3 Tee mullen iku sovindo,
4 Vanno vaka el. vankka veljexesi,
5 Vasten syöntäni sylöäk,
6 5 Vannottele vahtoani.

*Sedan beskrifves Myrjärnets uphof:*

8 Etpäs silloin suuri ollut,
9 Etsäs suuri etsäs pieni,
10 Et kovin koriakana,
11 Kuins sas suosta sotkettihin,
12 10 Vetelästä vellottihin.
13 Voi sinua Rauta raukka,
14 Rauta raukka koito kuona,
15 Teräs Tenho päivällinen,
16 Poikasi teki pahoa,
17 15 Lapsesi tihoa työtä,
18 Tule työsi tuntemahan,
19 Vikasi parantamaan.

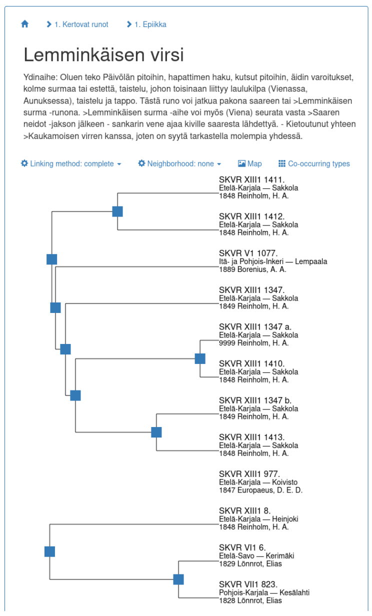
Similar poems

| | |
|---|---|
| SKVR IX4 825. | 29 % |
| SKVR XII1 4159. | 28 % |
| SKVR XV 240b. | 25 % |
| SKVR VII3 loitsut 445. | 24 % |
| SKVR XII1 4112. | 23 % |
| SKVR XII1 4150. | 22 % |



| SKVR XII1 4079. | SKVR IX4 825. |
|---|---|
| skvr12140790 | skvr09408250 |
| Pohjois-Pohjanmaa | Keski-Suomi — Petäjävesi |
| 1789 Ganander, Christfrid | 1851 Regnell, K. V. |
| **COL:** Ganander | **COL:** Regnell |
| **ID:** 4079. | **ID:** 825. |
| **INF:** | **INF:** Koppelniemi. Vanha muori.#1 |
| **LOC:** [Pohjanmaa.] | **LOC:** Petäjäv. |
| **OSA:** XII1 | **OSA:** IX4 |
| **SGN:** Myth. fenn. s. 77. | **SGN:** 6. |
| **TMP:** -1789. | **TMP:** 51. |
| 6. Loitsut | 6. Loitsut |
| ├ 05. Tautiloitsut | ├ 05. Tautiloitsut |
| │ └ Raudan sanat | │ └ Raudan sanat |
| └ 16. Syntyloitsut | └ 16. Syntyloitsut |
| └ Raudan synty | └ Raudan synty |
| | Oppa sinuas'#2 rauta kulta, |
| Rauta Rehki, kulta kilpi, | [Rauta kulta,]#3 Kulta kilpi, |
| Tee mullen iku sovindo, | |
| Vanno vaka el. vankka veljexesi, | |
| Vasten syöntäni sylöäk, | |
| 5 Vannottele vahtoani. | |
| Etpäs silloin suuri ollut, | Etväs sillo(i)n#4 suur' ollut, |
| Etsäs suuri etsäs pieni, | [Etväs suuri]#3 Etkä pieni, |
| Et kovin koriakana, | 5_Etkä kovin koreakaan, |
| | Kuin' sä#5 maitona makasit |
| | Nuoren neittosen nisissä, |
| Kuins_sas suosta sotkettihin, | Kuin_sinua suossa sullottihin |
| 10_Vetelästä vellottihin. | Vetelässä vellottihin_-_-_- |
| Voi sinua Rauta raukka, | |
| Rauta raukka koito kuona, | |
| Teräs Tenho päivällinen, | |
| Poikasi teki pahoa, | |
| 15 Lapsesi tihoa työtä, | |
| Tule työsi tuntemahan, | |
| Vikasi parantamaan. | - - - |

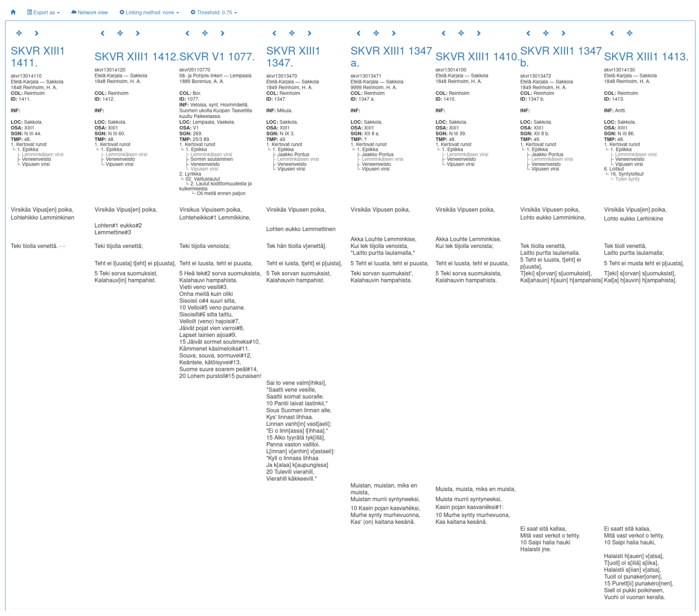
Figure 5: The Runoregi user interface showing a single poem with a list of similar poems (left) and an alignment of two similar poems (right). Note: the images were edited by removing or rearranging some UI elements to simplify the presentation.



Lemminkäisen virsi

Ydinaihe: Oluen teko Päivölän pitoihin, hapattimen haku, kutsut pitoihin, äidin varoitukset, kolme surmaa tai estettä, taistelu, johon toisinaan liittyy laulukilpa (Vienassa, Aunuksessa), taistelu ja tappo. Tästä runo voi jatkua pakona saareen tai >Lemminkäisen surma -runona. >Lemminkäisen surma -aihe voi myös (Viena) seurata vasta >Saaren neidot -jakson jälkeen - sankarin vene ajaa kiville saaresta lähdettyä. - Kietoutunut yhteen >Kaukamoisen virren kanssa, joten on syytä tarkastella molempia yhdessä.

Linking method: complete
Neighborhood: none
Map
Co-occurring types
SKVR XIII1 1411. Etelä-Karjala — Sakkola 1848 Reinholm, H. A.
SKVR XIII1 1412. Etelä-Karjala — Sakkola 1848 Reinholm, H. A.
SKVR V1 1077. Itä- ja Pohjois-Inkeri — Lempaala 1889 Borenius, A. A.
SKVR XIII1 1347. Etelä-Karjala — Sakkola 1849 Reinholm, H. A.
SKVR XIII1 1347 a. Etelä-Karjala — Sakkola 9999 Reinholm, H. A.
SKVR XIII1 1410. Etelä-Karjala — Sakkola 1848 Reinholm, H. A.
SKVR XIII1 1347 b. Etelä-Karjala — Sakkola 1849 Reinholm, H. A.
SKVR XIII1 1413. Etelä-Karjala — Sakkola 1848 Reinholm, H. A.
SKVR XIII1 977. Etelä-Karjala — Koivisto 1847 Europaeus, D. E. D.
SKVR XIII1 8. Etelä-Karjala — Heinjoki 1848 Reinholm, H. A.
SKVR VI1 6. Etelä-Savo — Kerimäki 1829 Lönnrot, Elias
SKVR VII1 823. Pohjois-Karjala — Kesälahti 1828 Lönnrot, Elias



Export as
Network view
Linking method: none
Threshold: 0.75
SKVR XIII1 1411.
SKVR XIII1 1412.
SKVR V1 1077.
SKVR XIII1 1347.
SKVR XIII1 1347 a.
SKVR XIII1 1410.
SKVR XIII1 1347 b.
SKVR XIII1 1413.

Figure 6: Dendrogram (left) and multi-poem alignment (right) views in Runoregi.

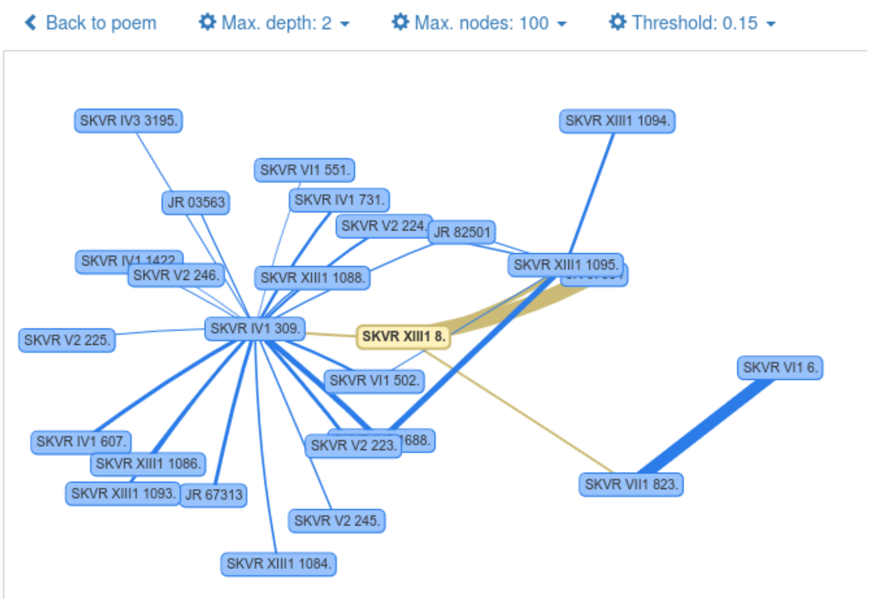

Back to poem    Max. depth: 2    Max. nodes: 100    Threshold: 0.15

Figure 7: The poem network view in Runoregi showing breadth-first search starting from the poem SKVR XIII1 8 (in yellow).

*Pieni Runon-seppä* to the national epic *Kalevala*. These works were based on oral poetry, but already shortly after their publication they became highly influential as fragments of them were learned by heart and reintroduced to the oral tradition. By studying the poem similarities and alignments from the periods both before and after the publication of a certain work we can gain new insights into this oral-literary-oral circuit.

For example, Fig. 8 shows the areal distribution of poems similar to the poem *Elkää sanoko huolettomaksi* (*Do not say [I] have no worries*) in Elias Lönnrot's *Kanteletar* (number 52 in book 1). The map on the left side shows poems collected before the publication of *Kanteletar* (1840), while the map on the right poems collected in 1840 and later. Caution is required in interpreting such maps: they do not necessarily show that the poem has spread to a wider area,

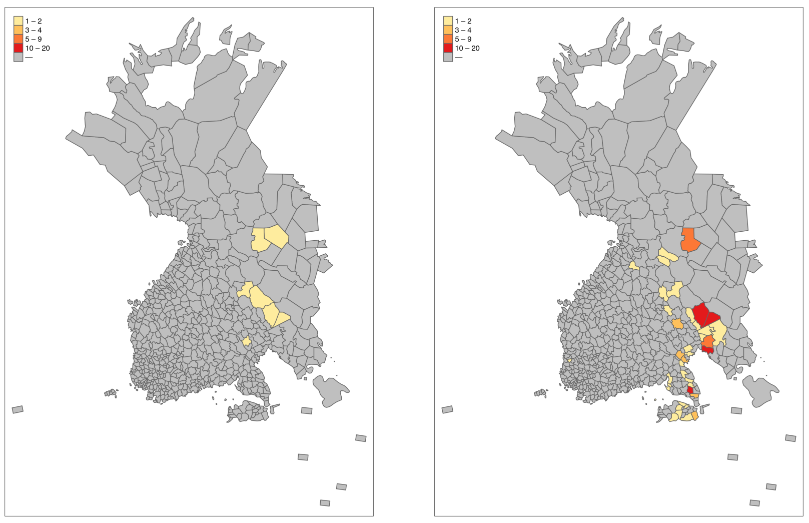


Figure 8: The areal distribution of poems similar to Kanteletar 1:52 *Elkää sanoko huolettomaksi* before 1840 (left) vs. 1840 and after (right).

as the tradition of some areas (notably Ingria) was only collected at a later period. However, this example shows how aggregate views and automatically computed similarities help to navigate through the collections and select relevant material for close reading.

Similar research has earlier been done manualy, mostly with focus on prominent literary works. Notably, Väinö Kaukonen traced the sources and earlier versions of Elias Lönnrot's works Kalevala and Kanteletar verse by verse [Kaukonen, 1956, 1984] [see also Hämäläinen, 2020]. A question of interest for folkloristic research is for example, how many lines or what kinds of story patterns did individual prominent singers contribute to the published works.

**Northern-Southern Finnic commonalities.** The collection and study of oral poetry has usually been done within national traditions (Finnish and Estonian). Thus, little is known about the commonalities of the Northern Finnic (Finnish, Karelian, Ingrian) and the Southern Finnic (Estonian, South Estonian) language areas. While the discovery of such similarities is especially difficult due to linguistic and orthographic differences, the ability of the algorithm to try the alignment of all poem pairs and work with low similarity thresholds allows us to detect similarities crossing the language boundary. Table 1 provides an example of such result: a fragment of the song *The maid to be ransomed* (*Lunastettava neito* / *Lunastatav neiu*) in Ingrian-Finnish and Estonian versions, with very similar structure indicating a close relationship. The rightmost column shows the bigram-based verse similarity (non-rescaled), with values above the 0.5 threshold in bold. The similarity of those pairs can be thus detected automatically.

| **Ingrian-Finnish** | **Estonian** | **translation** | **sim.** |
|---|---|---|---|
| Lilla istu kamperissa, | Lilla istus kammeris, | The girl was sitting in a chamber, | **.79** |
| Aik' oli ikäv uottaa, | Tal aeg oli igav oota. | It was a sad time waiting. | .46 |
| Näki vennan reissivanna | Ta nägi venda sõudema | She saw a brother [travelling / rowing] | .20 |
| Pitkin mere rantaa. | Seal üle mereranna. | Along the sea coast. | .45 |
| "Rikas venna, rakas venna, | "Kulla venda, rikas venda | 'Rich brother, [dear / golden] brother | **.64** |
| Lunast minnuu täältä vällää!" | Lunasta mu südant!" | Ransom [me from here / my heart]!' | .31 |
| "Millä mie lunassan, | "Kellega ma lunastan, | 'With what do I ransom you, | .41 |
| Kui miull' ei ole varraa?" | Kui mul ei ole raha." | When I don't have money?' | **.73** |
| "On siull' koton kolme miekkaa, | "Sul on kodu kolmi mõeka, | 'You've got three swords at home, | **.66** |
| Pane niist' yksi pantiks!" | Pane üks neist pandiks." | Pawn one of them!' | **.74** |
| "Enne mie luovun siusta | "Ennem mina lahkun õekesest, | 'I'd rather give up [you / a sister], | .36 |
| Kui omast' kolmest' miekast'." | Kui oma sõjamõegast." | Than my own [three / war] sword[s].' | .44 |

Table 1: Fragment of an Ingrian-Finnish and Estonian version of the song *The maid to be ransomed*, showing the possibility of cross-lingual alignment.

## VI CONCLUSION

In this article I presented an optimized version of the weighted sequence alignment algorithm developed for the purpose of comparing texts in large collections of oral folk poetry. The algorithm is based on the well-known 'weighted edit distance' algorithm, but reformulated in terms of vector and matrix operations, which allows it to operate on a large number of sequences (here: texts) simultaneously. This allowed us to compute alignment-based similarity scores for each pair of texts within a corpus of more than 280,000 items – a task that would not be possible to accomplish with the standard pairwise sequence comparison. The resulting similarity table can serve as a basis for a quantitative study of text similarity Finnic oral folk poetry from various angles, some of which were briefly sketched.

## FUNDING

This work was funded by the Academy of Finland research project no. 333138 'Formulaic intertextuality, thematic networks and poetic variation across regional cultures of Finnic oral

poetry'.

## References

Thomas Bocek, Ela Hunt, and Burkhard Stiller. Fast similarity search in large dictionaries. Technical report, University of Zurich, 2007.

Neil Coffee, Jean-Pierre Koenig, Shakthi Poornima, Christopher W. Forstall, Roelant Ossewaarde, and Sarah L. Jacobson. The tesserae project: intertextual analysis of latin poetry. *Literary and Linguistic Computing*, 28(2), 2013.

Christopher W. Forstall and Walter J. Scheirer. *Quantitative intertextuality: analyzing the markers of information reuse*. Springer, 2019.

Lauri Honko. Text as process and practice: the textualization of oral epics. In Lauri Honko, editor, *Textualization of Oral Epics*, pages 3–54. De Gruyter Mouton, 2000.

Nina Hämäläinen. Säe säkeeltä: Väinö Kaukosen säetutkimukset kalevalasta. *Sananjalka*, 62:215–235, 2020.

Maciej Janicki. Optimizing the weighted sequence alignment algorithm for large-scale text similarity computation. In *Proceedings of the 2nd International Workshop on Natural Language Processing for Digital Humanities*, pages 96–100, Taipei, Taiwan, November 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.nlp4dh-1.13.

Maciej Janicki, Kati Kallio, and Mari Sarv. Exploring Finnic oral folk poetry through string similarity. *Digital Scholarship in the Humanities*, 38(1):180–194, 2023. URL https://doi.org/10.1093/llc/fqac034.

Stefan Jänicke and David Joseph Wrisley. Visualizing mouvance: Toward a visual analysis of variant medieval text traditions. *Digital Scholarship in the Humanities*, 32:ii106–ii123, 2017.

Kati Kallio, Maciej Janicki, Eetu Mäkelä, Jukka Saarinen, Liina Saarlo, and Mari Sarv. Eteneminen omalla vastuulla. Lähdekriittinen laskennallinen näkökulma sähköisiin kansanrunoaineistoihin. *Elore*, 30(1):59–90, 2023. URL https://doi.org/10.30666/elore.126008.

Väinö Kaukonen. *Elias Lönnrotin Kalevalan toinen painos*. Suomalaisen Kirjallisuuden Seura, Helsinki, 1956.

Väinö Kaukonen. *Elias Lönnrotin Kanteletar*. Suomalaisen Kirjallisuuden Seura, Helsinki, 1984.

Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

David Rosson, Eetu Mäkelä, Ville Vaara, Ananth Mahadevan, Yann Ryan, and Mikko Tolonen. Reception reader: Exploring text reuse in early modern british publications. *Journal of Open Humanities Data*, 9(5):1–11, 2023. DOI: https://doi.org/10.5334/johd.101.

Susan Schreibman, Amit Kumar, and Jarom McDonald. The versioning machine. *Literary and Linguistic Computing*, 18(1):101–107, 2003.

Klaus Schulz and Stoyan Mihov. Fast string correction with levenshtein-automata. *International Journal of Document Analysis and Recognition*, 5:67–85, 2002.

Tommaso Soru and Axel-Cyrille Ngonga Ngomo. Rapid execution of weighted edit distances. In *Proceedings of the 8th International Workshop on Ontology Matching*, 2013.

Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(I): 168–173, 1974.