Detecting and Analyzing Text Reuse with BLAST

Master's Thesis
University of Turku
Department of Future Technologies
2018
Aleksi Vesanto

UNIVERSITY OF TURKU
Department of Future Technologies

ALEKSI VESANTO: Detecting and Analyzing Text Reuse with BLAST

Master's Thesis, 54 p., 13 app. p.
TurkuNLP Group
December 2018

Tässä tutkielmassa jatkan edellistä työtäni tekstin uudelleenkäytön tunnistuksessa. Ehdotan uutta tekstintunnistus-menetelmä, joka käyttää apunaan BLAST:ia (Basic Local Alignment Tool). BLAST on algoritmi, joka on alunperin suunniteltu biolääketieteellisten sekvenssien, kuten DNA:n ja proteeiinisekvenssien vertailuun ja linjaukseen.

Käyn läpi alkuperäisen BLAST-algoritmin tarkasti askel askeleelta. Kuvailen myös kaksi muuta suosittua sekvenssilinjaus-menetelmää. Osoitan BLAST-pohjaisen tekstin uudelleenkäytön tunnistus -algoritmin tehokkuuden vertaamalla sitä viimeisimpään vastaavaan menetelmään ja näytän miten ehdottamani menetelmä toimii huomattavasti paremmin.

Käytän metelmää suomalaisten sanoma- ja aikakauslehtien korpuksen analysointiin, joka koostuu yli kolmesta miljoonasta skannatusta sivusta, jotka on muunnettu tekstiksi käyttäen OCR-ohjelmistoa (Optical Character Recognition). Luokittelen tulokset kolmeen kategoriaan: päivittäinen ja pitkäaikainen tekstin uudelleenkäyttö sekä viraalit uutiset. Kuvailen kategoriat ja annan niistä esimerkkejä sekä ehdotan samalla uutta menetelmää viraaliteetin laskemiseen klustereille.

Keywords: Tekstin uudelleenkäyttö, luonnollisen kielen käsittely, bioinformatiikka, OCR

UNIVERSITY OF TURKU
Department of Future Technologies

ALEKSI VESANTO: Detecting and Analyzing Text Reuse with BLAST

Master's Thesis, 54 p., 13 app. p.
TurkuNLP Group
December 2018

In this thesis I expand upon my previous work on text reuse detection. I propose a novel method of detecting text reuse by leveraging BLAST (Basic Local Alignment Search Tool), an algorithm originally designed for aligning and comparing biomedical sequences, such as DNA and protein sequences.

I explain the original BLAST algorithm in depth by going through it step-by-step. I also describe two other popular sequence alignment methods. I demonstrate the effectiveness of the BLAST text reuse detection method by comparing it against the previous state-of-the-art and show that the proposed method beats it by a large margin.

I apply the method to a dataset of 3 million documents of scanned Finnish newspapers and journals, which have been turned into text using OCR (Optical Character Recognition) software. I categorize the results from the method into three categories: every day text reuse, long term reuse and viral news. I describe them and provide examples of them as well as propose a new, novel method of calculating a virality score for the clusters.

Keywords: Text reuse, NLP, Bioinformatics, sequence alignment, OCR

TABLE OF CONTENTS

1 Introduction

Text reuse detection is an often needed task in several different applications that vary from pure research purposes to industrial cases. Seeing how news articles have circulated in the past opens up new and interesting research questions. On the other hand, it can be used to detect plagiarism, which is a very real issue, especially in academia. It can also be used to detect where a particular rumor online first occurred, which can be helpful to discredit so called ”fake news”. Text reuse detection is therefore not a new idea and several studies have already explored the possibility of it with varying success.

In this thesis I expand upon my work from the articles ”A System for Identifying and Exploring Text Repetition in Large Historical Document Corpora” (Vesanto et al., 2017b) and ”Applying BLAST to Text Reuse Detection in Finnish Newspapers and Journals, 1771-1910” (Vesanto et al., 2017a). In these articles I detail a new, novel approach to detecting text reuse, which achieves state-of-the-art results in both precision and recall. Therefore in this thesis, I go through all the necessary steps to fully understand how the system functions as well as new modifications made to the algorithm since the publication of the two articles.

The software finds text reuse by leveraging an old, yet popular algorithm, Basic Local Alignment Search Tool (BLAST). The algorithm is originally designed for comparing and aligning biomedical sequences, such as DNA and protein sequences. I explain step-by-

step how the algorithm works when aligning protein sequences. As all implementations of the algorithm assume that they are used for aligning biomedical sequences, they can make assumptions about the alphabet being used. That is, it is limited by the number of unique amino acids available.

As the implementations of BLAST make assumptions that do not hold when aligning natural language, certain pre-processing steps are necessary to be able to align natural language with it. I explain how I pre-process the texts by encoding them into protein sequences, which then can be processed by BLAST. The output from it is then clustered, so that chains of similar sequences can subsequently be extracted. These chains thus contain all occurrences of a particular text reuse passage. Using BLAST as the back-end does come with certain caveats. I explain what these are and provide methods to overcome these.

BLAST uses a substitution matrix for scoring when performing the alignment between two sequences. I explain how these work in depth and also propose a way to calculate a custom, data specific substitution matrix, which improves the precision and recall of the reuse detection.

I demonstrate the viability of the reuse detection algorithm by applying it to the entire dataset of Finnish newspapers and journals from 1771 to 1910. These are scanned documents which have been then OCR read (Optical Character Recognition) into text. The image-to-text process contains several issues, such as bad scans, leaked ink, unoptimal OCR system as well as the hard to read font, Fraktur, which was prevalent in the papers at the time. These issues together cause the OCR quality to be, at times, extremely erroneous. I show how the BLAST algorithm, even through the massive OCR noise, can detect large quantities of text reuse clusters from the data.

To see how the system performs compared to other readily available text reuse detection methods, I create a subset of documents from the corpus and compare the precision and recall of both BLAST and Passim, the previous state-of-the-art system. I show that BLAST outperforms Passim by a massive margin.

Lastly, I analyze the found text reuse clusters from the corpus. I categorize them into three overlapping categories: every day reuse, long term reuse and viral news. I analyze each of these and provide an explanation of what they are or what they can be used for. I also propose a novel way of calculating a virality score to rank the clusters based on their viralness, which is vital when attempting to observe what kind of articles went viral in the past.

2 Related work

Text reuse detection can be thought of as segmented text document similarity calculation. The process entails detecting similar documents and then clustering the similar documents so that a single cluster contains all occurrences of a particular text reuse. It therefore needs an accurate enough method to detect similarity and a robust clustering method. Many different measures for these purposes have been proposed and used in the natural language processing field.

Document similarity calculations go all the way back to 1960s, where rudimentary document similarity was used in information retrieval to automatically retrieve documents relevant to a given query (Salton and Lesk, 1968). Text similarity measures have been used in different tasks than just retrieving documents. (Hall and Dowling, 1980) leveraged text similarity to approximate string matching, where two strings are very similar, but have spelling errors or other slight variations that separate them from each other. By calculating text similarity between the strings, they could match strings that were nearly identical.

Text similarity has been used as a feature for clustering. (Huang, 2008) used it to cluster similar documents into clusters. He formed word frequency vectors for all of the documents and used the cosine distance of two document vectors to measure their similarity. Two very similar documents would then be clustered into the same cluster by the cluster-

ing algorithm. Computing similarities of just thousands of documents can be done easily, but back in 2008, when the size of the corpus rises to millions, a single computer might not have had enough computing power to handle the necessary computations. (Elsayed et al., 2008) performed pairwise document similarity detection using a novel method, where they could efficiently compute the similarities in a large corpus of over 900,000 documents, by reducing the document vectors into smaller sub vectors and then parallelizing the computation over several computers. This allowed them to complete their task in reasonable amount of time.

Text reuse detection is often used to see how certain texts change or spread over time. For this purpose, it is necessary to chain, or cluster, together the similar documents. (Metzler et al., 2005) measured the information flow in TREC (Text Retrieval Conference) corpus. They performed both sentence and document level similarity measurements to cluster similar texts. Once they had all occurrences of a particular passage detected, they used the included meta data from the documents to see how the information spreads within the corpus. In similar fashion, (Bamman et al., 2017) tackled the problem of estimating the first publication of a composition. They had to find the chain of publications for each separate entity, similar to seeing how news may travel over time. They achieved this by looking at the meta data as well as the text of the compositions themselves and trained a classifier using manually annotated data to detect similar compositions for the clustering.

A lot of different rumors and ”facts” circulate all over the internet, yet often no one really knows where they originate. (Bendersky and Croft, 2009) attempted to detect these chains online by first using normal information retrieval methods to find possible candidate documents and then using mixture of word overlap and query likelihood to measure similarity. Once they have found all cases of a certain entity, a rumor for example, they could sort them by their dates and inspect the first occurrence to see where it originated

from.

Actual text reuse detection is in no way a recent idea, (Clough et al., 2002) attempted to detect it by looking at three different criteria between documents: ngram overlap, greedy string tiling and sentence alignment. In practice, for every document pair they looked at the fraction of shared ngrams, the degree of substring similarity and aligned derivations of the original text. To estimate their results, they manually annotated a corpus of newspapers on multiple levels. In their corpus they annotated roughly 400 complete or partially derived articles down to lexical level, where for each word or phrase they knew whether it has appeared in the original verbatim or slightly rewritten (Gaizauskas et al., 2001).

Sometimes the used data can be extremely old. (Lee, 2007) described a method they used to detect text reuse in Greek New Testament. Their goal was to extract a verse from a target text and determine whether it is derived from a verse in a source text. To this end, they used the cosine distance of the TF-IDF vectors created from the target and source verses. The vectors thus contain the term frequency for all tokens, which are weighted using the inverse document frequency of the respective token. Any token that appears regularly throughout the whole corpus is thus deemed to be irrelevant. They also used data specific heuristics to extract extra features. Due to the small corpus size, they wanted to limit their feature space as much as possible. Therefore in total they only used one feature, the cosine distance, in addition to the heuristic features.

Several competitions for text reuse detection have also been organized. At PAN 2014 conference, several teams competed against each other in a task focusing on plagiarism detection, which can be thought of as text reuse detection. In the task they had to identify all contiguous passages between two documents that contained text reuse. Several different methods were employed by the teams, but the winning team achieved the highest score

by applying several different methods together. They first pre-processed the texts using heuristics, then used seeding to determine possible candidates for aligning, expanded the alignment from the seed tokens to detect the contiguous passage and then filtered out incorrect passages. These methods together were successful enough to beat other teams. (Sánchez-Pérez et al., 2014)

Several other studies have explored the possibility of plagiarism detection, which is a relevant task, as nowadays it is easy to get access to large amount of texts from large variety of subjects. It is therefore easy to find content to plagiarize. A single person can not be assumed to have the time to read and memorize all of them. Therefore any plagiarism detection system needs to be automatic. Different software are used nowadays in classes, such as Turnitin, which compares the document against its massive database of published and authored content. A study using Turnitin showed that over 60% of students of University of Botswana had a low-scale plagiarism in their essays. Only 14% had legitimate research, whereas as almost 9% had high-scale plagiarism, meaning that they plagiarized at least one paragraph almost completely. (Batane, 2010). This shows that plagiarism is a very real issue.

Most plagiarism software are used to check whether new to-be-published articles contain plagiarism. (Citron and Ginsparg, 2015) conducted a thorough study to see how much content was plagiarized in scientific articles by looking at all articles from arXiv.org from 2012 to all the way back to 1991. They used 7-grams to detect reuse, while filtering out common phrases that appear a lot in most scientific articles. They show that plagiarism has happened before and is still happening. They also show that submissions from less developed countries tend to contain more plagiarism than submissions from more developed ones. Certain authors are also more prolific in plagiarism. In the articles examined, certain authors had over a hundred submissions flagged as being plagiarized. (Gupta and

Rosso, 2012) performed a similar study, but in it they only focused on a single, large, conference, ACL (Association for Computational Linguistics). They used an open source software WCopyFind to detect plagiarism, which does it by comparing hashed values of the tokens in the text. They showed that reuse between articles is on the rise, particularly between two long papers (8 or more pages). Plagiarism is increasing in short papers as well, but not as extensively.

While there exists many different approaches to plagiarism detection, many of them can be fairly easily fooled by merely changing the word order or using synonyms. (Potthast et al., 2010) proposed a framework to evaluate plagiarism, which tries to overcome the shortcomings of previous plagiarism detection methods. Many people plagiarize and try to obfuscate it to fool both automatic and manual verification systems. Their framework tries to combat this by automatically generating obfuscated versions of the text, which they showed is a valid method to mimic manual obfuscation. The framework can then evaluate whether a plagiarism detection method is robust enough or not.

Most of the previous studies either have a defined article segmentation or some other similar assumption. This allows them to merely use simple document similarity measures. In many cases though, there is no segmentation available, and the text reuse is embedded into otherwise different content. This causes issues when calculating the similarity, as then the dissimilar segments of the documents heavily penalize the calculated similarity. (Smith et al., 2014) developed a software to model text reuse from within a document with several different subjects. They successfully applied this to a 20th century American newspaper corpus and extracted clusters of text reuse, with the goal of detecting which news became viral (Smith et al., 2015). Their software, Passim, can detect text reuse well when the OCR quality is adequate, i.e. there are not too many errors. Using their software, they attempted to model infectious texts and analyzed how they spread ge-

ographically as well as see how different publications are linked together and what kind of time dynamics were in play at the time (Smith et al., 2013).

3 Basic Local Alignment Search Tool (BLAST)

Sequence alignment algorithms can be separated into two categories: global and local alignment algorithms. The first are algorithms that optimize the complete alignment between two documents. These algorithms are akin to normal document similarity methods, as they tell how similar the two documents are globally. The latter algorithms attempt to find only the similar regions between the documents. That is, the alignment of two documents may yield several different regions that are highly similar between the documents. This means that regions that are clearly different are not considered when determining the similarity of a document pair.

BLAST (Basic Local Alignment Search Tool) is an old yet efficient algorithm used to compare biomedical sequences, such as DNA and protein sequences. (Altschul et al., 1990). It was first published in 1990, and at the writing of this thesis, it has been cited over 70,000 times. While the algorithm is old, the implementation of it has been continuously improved. In this thesis I use NCBI BLAST software as the implementation, and more specifically the protein version, BLASTP. (McGinnis and Madden, 2004) Thus, I explain how the original algorithm works in regard to dealing with protein sequences.

The core idea of BLAST is to compare and align query sequences against a massive

database of protein sequences. The algorithm is designed to be capable of aligning query sequences against massive databases, such as different genomes, while still retaining a relatively small processing time. The algorithm consists of multiple different steps it has to go through when querying a sequence against the database. (Altschul et al., 1990) Other approaches for the same task exist, but they are either less accurate or are not designed to handle such massive databases. (Waterman et al., 1976)

In this chapter I explain how BLAST performs local alignment step-by-step and how its scoring works using a substitution matrix. I also describe two alignment algorithms, a local and a global one, which BLAST uses in the algorithm and I use later in the thesis, respectively.

3.1 Substitution matrix

When comparing two sequences, or two amino acids in the sequences to be more specific, the substitution matrix informs the algorithm how much to reward a matching amino acid or penalize a mismatch. Using this matrix a value can be calculated that states how much two regions of sequences look like one another. It is thus a matrix with a shape $N * N$, where N is the amount of amino acids. In NCBI BLAST this is 25. 23 amino acids and partials and then a couple BLAST specific extras. (McGinnis and Madden, 2004) The idea behind such matrix is to be able to give different rewards and penalties depending on the amino acids.

In bioinformatics there are multiple different pre-calculated substitution matrices that have been proven to help the scoring, such as BLOSUM62. This is because certain amino acids can be interchangeable or at least closer to each other, so a sequence does not have to be exact match for it to be relevant. (Eddy, 2004) Example of BLOSUM62 can be

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	4	0	-2	-1	-2	0	-2	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-2	-3	-2
C	0	9	-3	-4	-2	-3	-3	-1	-3	-1	-1	-3	-3	-3	-3	-1	-1	-1	-2	-2
D	-2	-3	6	2	-3	-1	-1	-3	-1	-4	-3	1	-1	0	-2	0	1	-3	-4	-3
E	-1	-4	2	5	-3	-2	0	-3	1	-3	-2	0	-1	2	0	0	0	-3	-3	-2
F	-2	-2	-3	-3	6	-3	-1	0	-3	0	0	-3	-4	-3	-3	-2	-2	-1	1	3
G	0	-3	-1	-2	-3	6	-2	-4	-2	-4	-3	-2	-2	-2	-2	0	1	0	-2	-3
H	-2	-3	1	0	-1	-2	8	-3	-1	-3	-2	1	-2	0	0	-1	0	-2	-2	2
I	-1	-1	-3	-3	0	-4	-3	4	-3	2	1	-3	-3	-3	-3	-2	-2	1	-3	-1
K	-1	-3	-1	1	-3	-2	-1	-3	5	-2	-1	0	-1	1	2	0	0	-3	-3	-2
L	-1	-1	-4	-3	0	-4	-3	2	-2	4	2	-3	-3	-2	-2	-2	-2	3	-2	-1
M	-1	-1	-3	-2	0	-3	-2	1	-1	2	5	-2	-2	0	-1	-1	-1	-2	-1	-1
N	-2	-3	1	0	-3	0	-1	-3	0	-3	-2	6	-2	0	0	1	0	-3	-4	-2
P	-1	-3	-1	-1	-4	-2	-2	-3	-1	-3	-2	-1	7	-1	-2	-1	1	-2	-4	-3
Q	-1	-3	0	2	-3	-2	0	-3	1	-2	0	0	-1	5	1	0	0	-2	-2	-1
R	-1	-3	-2	0	-3	-2	0	-3	2	-2	-1	0	-2	1	5	-1	-1	-3	-3	-2
S	1	-1	0	0	-2	0	-1	-2	0	-2	-1	1	-1	0	-1	4	1	-2	-3	-2
T	-1	-1	1	0	-2	1	0	-2	0	-2	-1	0	1	0	-1	1	4	-2	-3	-2
V	0	-1	-3	-2	-1	-3	-3	3	-2	1	1	-3	-2	-2	-3	-2	-2	4	-3	-1
W	-3	-2	-4	-3	1	-2	-2	-3	-3	-2	-1	-4	-4	-2	-3	-3	-3	-3	11	2
Y	-2	-2	-3	-2	3	-3	2	-1	-2	-1	-1	-2	-3	-1	-2	-2	-2	-1	2	7

Figure 3.1: Example of the substitution matrix BLOSUM62.

seen in Figure 3.1. Of course using a such pre-compiled matrix does not make sense when not dealing with amino acids in a protein sequence. The probabilistic assumptions made in it do not hold with arbitrary alphabet. Thus when aligning natural language, a identity matrix is often used, where all matches are rewarded equally and mismatches penalized equally. The score for aligning two amino acids would then be evaluated using the equation:

$$score = S(a_1, a_2) \tag{3.1}$$

where S is the function that uses the substitution matrix and a_1 and a_2 are the two amino acids being scored. For example, if the used substitution matrix is BLOSUM62 and a_1 and a_2 are amino acids A and W, respectively, the score would be -3.

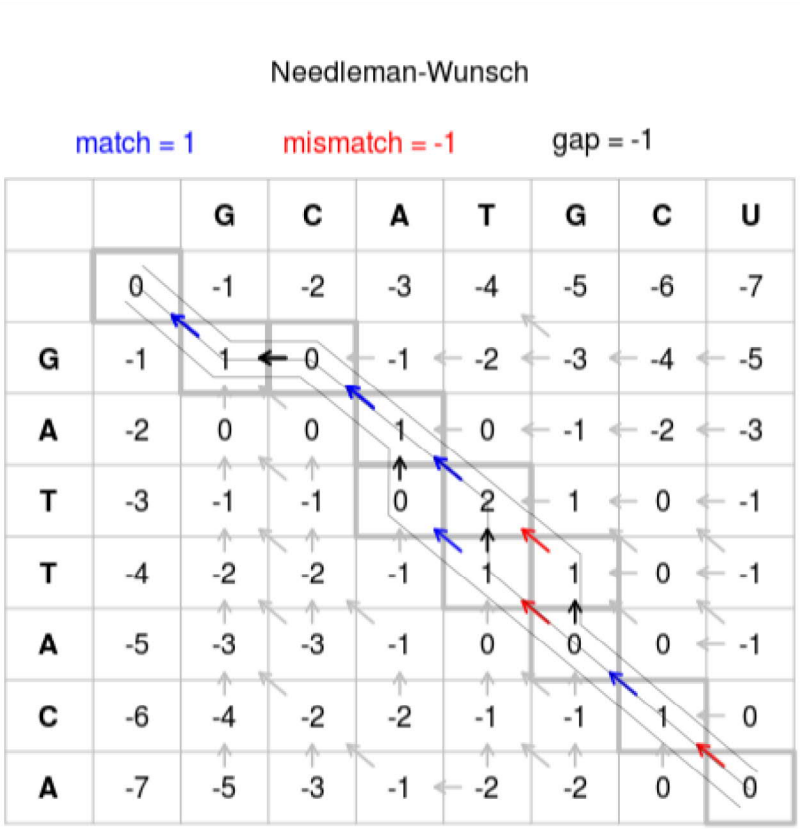

Figure 3.2: Example of the matrix after the Needle-Wunsch algorithm has found the optimal solution. (Commons, 2018a)

In section 5.1 I explain how I calculate a custom, data specific substitution matrix to improve the results when the data is natural language rather than protein sequences.

3.2 Needleman–Wunsch algorithm

Needleman-Wunsch algorithm is another old yet famous algorithm from the 1970s. It can be used to align any two sequences, though its primary purpose is to align biomedical sequences. (Needleman and Wunsch, 1970)

When aligning sequences s_1 and s_2, the algorithm starts with an empty matrix of size $(n+1) * (m+1)$, where n and m are the lengths of s_1 and s_2, respectively. The matrix

thus has a row and a column for every letter in the sequences plus one extra row and column. The core idea of the algorithm is to score matches and mismatches in the sequences. This scoring can be done by using a substitution matrix like explained in the previous section or just set values for match and mismatch. The next step is to fill the matrix. The first value $M_{0,0}$ is set to zero, as it naturally has no match or mismatch due to it being an extra row/column. Next, all other cell values are calculated row by row, using the equation:

$$M_{i,j} = max \begin{cases} M_{i-1,j-1} + s(a_i, b_j), \\ M_{i,j-1} + s(a_i, b_j), \\ M_{i-1,j} + s(a_i, b_j) \end{cases} \tag{3.2}$$

where $s(a_i, b_j)$ denotes the reward / penalty from the substitution matrix, when aligning the ith character from s_1 and jth character in s_2. The value is thus the maximum value of the scores of the cells that are to the left, on top or top-left diagonal of the current cell plus the reward / penalty value. The values in the first row and first column have only the left cell and top cell that they can use to calculate the score, so the values in those will just be the cumulative mismatch value. The first cell in the matrix that has all three neighboring values is at $M_{1,1}$, so its value will be the maximum of the three neighboring values plus the score value of $s(a_0, b_0)$.

In addition to just filling the matrix, a second, trace matrix is formed. This means that whenever picking one of the three neighboring values, the algorithm also remembers which one of them it picked. After the whole matrix is filled and traces are set, the algorithm finds the optimal alignment by tracing back to the origin from the bottom right cell using the directions from the trace matrix. If in the trace the picked direction was top-left, it denotes that there was either a match or a mismatch in the sequences. Left or top directions denote that there has been either an insertion (new character) or a deletion (missing character) in one of the sequences. This is a gap and is denoted by a ”-” character in the output. Following all the way back to $M_{0,0}$ in the top left will give the

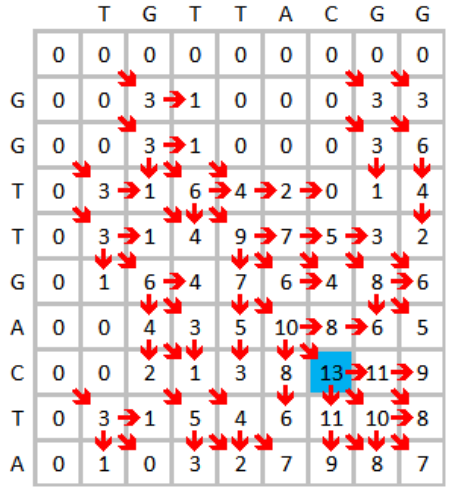

Figure 3.3: Example of the finished Smith-Waterman scoring matrix with the trace matrix overlayed on top. (Commons, 2018b)

best global alignment of the sequence. Example of this is shown in the Figure 3.2. In the example, two protein sequences, GATTACA and GCATGCU are aligned. Both the value matrix and the trace matrix are shown on top of each other, where the traces are denoted by arrows.

3.3 Smith-Waterman algorithm

Smith-Waterman algorithm is another popular sequence alignment algorithm. It is a modification of Needleman-Wunsch algorithm, where the main difference of it is that it is a local alignment algorithm instead of global. (Waterman et al., 1976)

In the same fashion as in the Needle-Wunsch algorithm, to align sequences $s1$ and $s2$, the first step is to initialize a matrix of size $(n + 1) * (m + 1)$, where n and m are the lengths of $s1$ and $s2$, respectively. The matrix is filled the same way as the Needle-Wunsch algorithm, but with a few tweaks. Firstly, the first row / column values are set to zero, rather than cumulative gap penalty values. No negative values are allowed in the

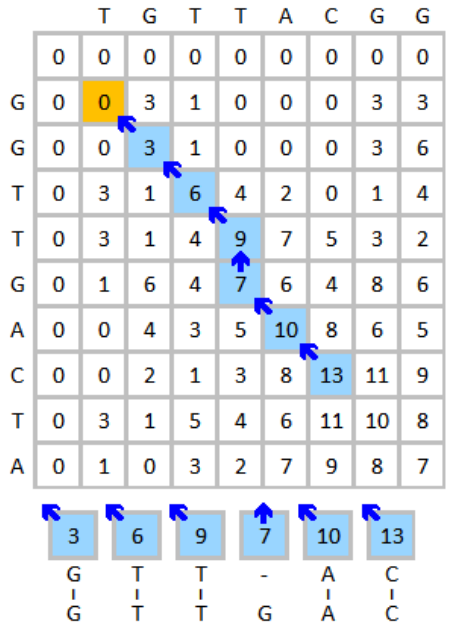

Figure 3.4: Example of the Smith-Waterman trace back process and the aligned results for the two sequences. (Commons, 2018c)

matrix, therefore any value that would otherwise get a negative value is set to zero. This is what allows the algorithm to find local alignments. Setting the cell value to zero means that prior similarities do not influence the possibility of a new, local similarity. In the filling process, the same trace matrix is also created and updated while filling the value matrix.

The third difference between the two algorithms is after the matrices are formed and the tracing back process begins. Rather than starting the alignment from the cell in the bottom right corner and continuing until the cell in the top left corner, the starting point is set to the cell with the highest value. There may be multiple cells that share the highest value. A trace back is started from all of these cells and continued until a zero is found. At any point in the trace back process a cell can be found with multiple equally possible directions in the trace matrix. This again branches the alignment, which all then need to be fully traced. Once all branches have been fully traced back to a zero, a score can

be calculated for all of them by summing their values in the trace. The one trace with the highest score is thus considered the most optimal alignment. Multiple traces may end up with the same score, which then causes them all to be considered equally optimal solutions.

3.4 BLAST algorithm

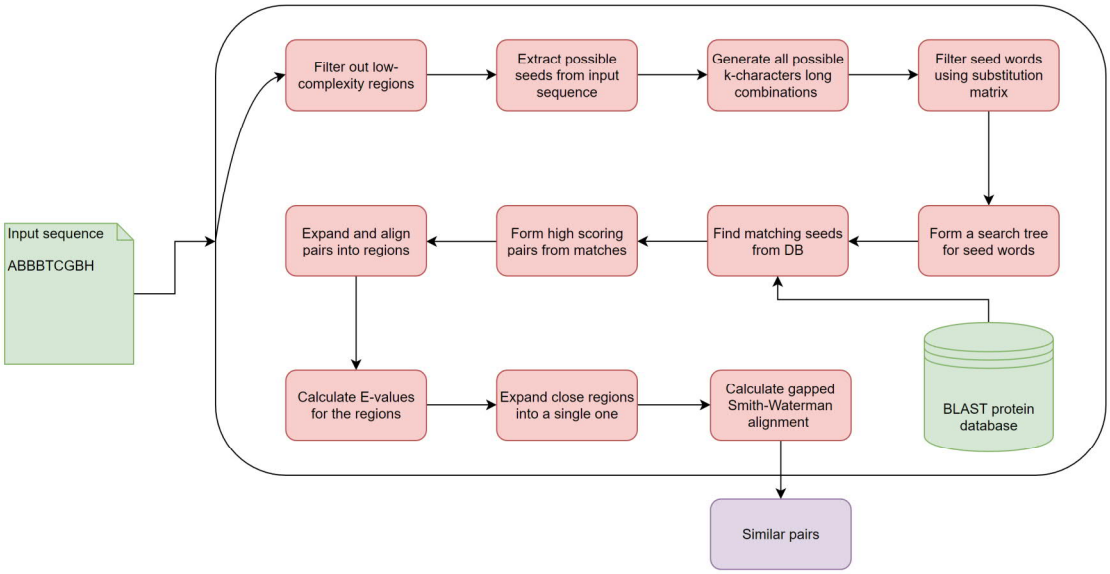

Figure 3.5: A diagram showing the BLAST algorithm.

The first step is to preprocess the input sequences by filtering out low-complexity regions from them. Low complexity regions consist of few different amino acids, e.g. ”AAAAAAAAAAAAAG”. They can also be short sub-sequences that appear regularly, e.g. a part of a amino acid sequence that is commonly shared by most sequences. In normal natural language sequences such regions could be common words or phrases, such as function words or even data specific words, e.g. title of a newspaper. These are irrelevant here and would only serve to confuse the algorithm in the later steps, where seed n-grams are extracted to filter out improbable candidate documents from the database to reduce unnecessary computations. (Mount, 2001) In NCBI BLAST, custom low-complexity re-

gions can be denoted to the algorithm by lowercasing letters in the desired region of the input sequence. (McGinnis and Madden, 2004)

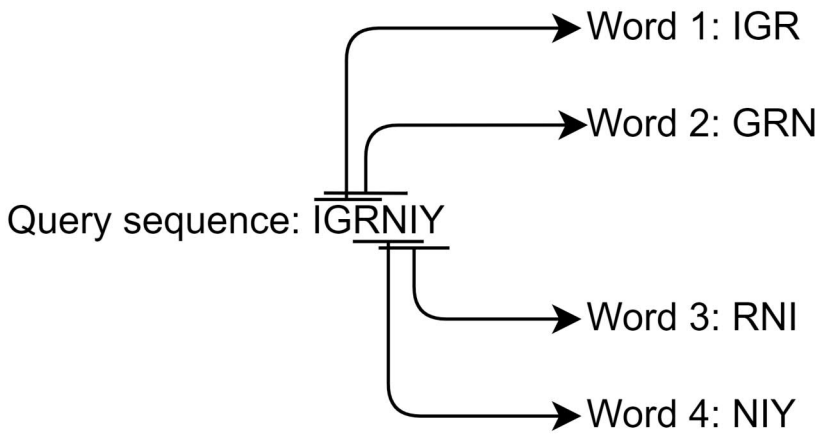

Figure 3.6: Example of how the seed words are extracted.

After the data has been pre-processed and filtered, the comparison phase begins. First a seed list is formed, where each seed is k characters long. Seeds are extracted sequentially from the input sequence, so two sequential seeds overlap by $k - 1$ characters. Example of such extraction can be seen in Figure 3.6. These seeds are used in the later steps to filter out sequences that do not share enough similarity between each other. Therefore the idea is to first look for simple n-gram similarities between the sequences to drastically lower the possible similar candidates. Setting the k values correctly is important, as having too high value might cause BLAST to miss matching similar sequences, e.g. if there is a lot noise. On the other hand, a k value too small would drastically increase the needed computing time, as almost all sequences share some parts, effectively disabling the filtering. (Mount, 2001) In NCBI BLAST, k can be between 2 to 7 characters long. (McGinnis and Madden, 2004)

BLAST achieves its relative fastness when querying a massive database by applying

clever filtering. In the next phase all possible k-character long seed combinations are generated and they are compared against the seeds extracted in the previous phase. The comparison between two seeds is done by weighting the comparison seed using a scoring matrix or a substitution matrix, as explained in chapter 3.1. In practice, the matrix has a match and mismatch values for each character in the seeds, so the seed pair gets a score based on the matches and mismatches between the seeds. This score is then compared against a set threshold T. Any seed pair with a score lower than T are ignored and the rest are considered to be a match. This method improves the accuracy, as now the seeds only have to be fuzzily similar rather than identical. Of course this method similarly to having a too low k values causes more computational needs to gain the accuracy. The threshold value is one of the hyperparameters that can be optimized to obtain better results and should be set according to the substitution matrix. It can also be set so high that no combination exceeds it, effectively disabling fuzzy search in favor of less computation power. (Mount, 2001)

The seed pairs with a score that exceeded the threshold are then organized into a efficient tree format for further analysis. This tree format makes it quicker for the algorithm to scan for documents with matching seeds. As all seeds get a score based on the substitution matrix, the algorithm can go through the tree until the score is too low based on the score threshold. Any child of that seed can automatically be discarded, as they have even lower score values. Example of such tree can be seen in Figure 3.7.

The next step is to scan through the whole sequence database and find every sequence that contains at least one of the high-scoring pairs evaluated in the previous step. If a match is found, that sequence will be a candidate for a comparison against the given query sequence. (Mount, 2001) Starting to compare the sequences if there is just one high-scoring pair would lead to massive amount of computations, especially if the k-value was low.

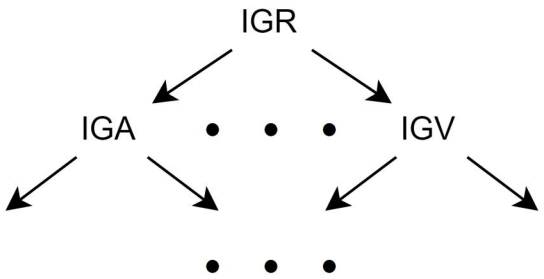

Figure 3.7: Example of the seed pairs in a tree format.

Therefore in NCBI BLAST, there is a window size parameter W, which specifies the size of a window inside which two seeds must be found. Two matches within a window define a region, which is a candidate for a similar subsequence. (McGinnis and Madden, 2004)

Once two pairs within a window have been found, the region is aligned, where the high-scoring pairs are then used as starting points in the aligning process. Just aligning the k-character long seeds is far from enough, therefore the high-scoring pairs are expanded. The algorithm expands the alignment in the query sequence in both directions, left and right. Each expanded character in the query sequence gets a score using the substitution matrix to reward matches and penalize mismatches. The score also takes in account the previous score, i.e. last left or right character score, depending on the expansion direction. This expansion is continued until the score reaches an empirically decided cutoff value C, such as 0. (Mount, 2001) Example of the expansion procedure can be seen in Figure 3.8.

The two seeds in the query sequence are not necessarily next to each other, especially if the window size is set to a large value. Therefore expanding these does not necessarily mean that they will be connected in the alignment, especially if the the region they are in is not actually similar. Therefore all score sequences of non-negatives are extracted from the alignment and a raw score S_R is calculated for them, by summing their scores

together. Raw score thus takes in account matches, mismatches and gaps. Example of the extraction is shown in Figure 3.8. If S_R is smaller than a score threshold S_T, that sequence is then discarded. This filters out alignments that are too short, based on the threshold value. Two seeds that overlap due to the expansion process are considered the same, as the chain contains both of them.

After filtering out expanded regions with too low score S_R, an expected value E_C is

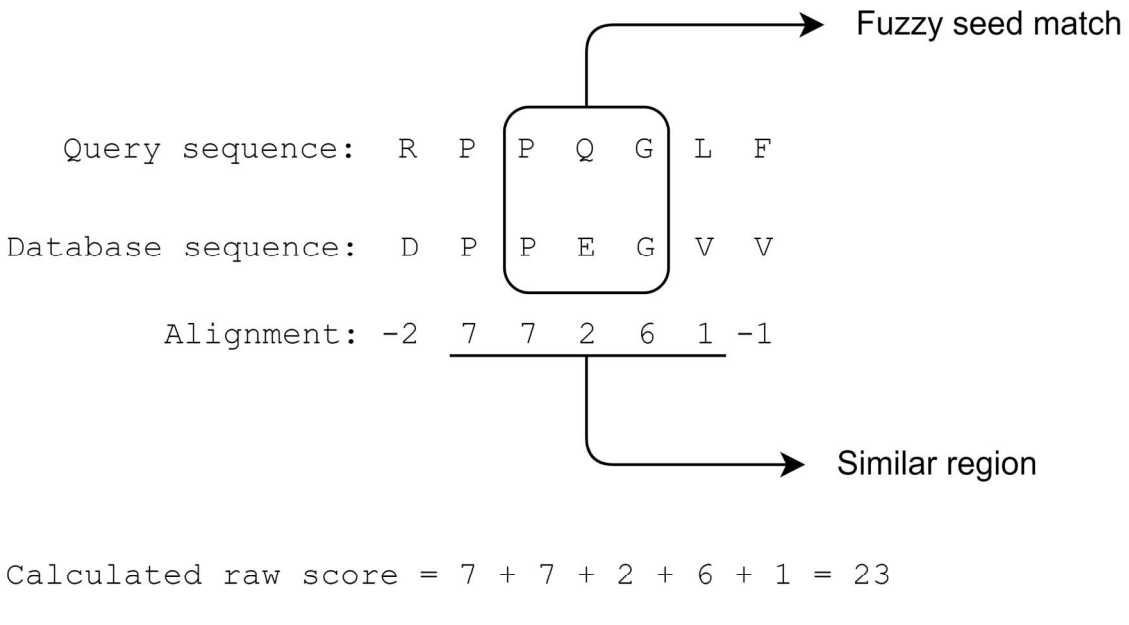

Figure 3.8: Example of the aligning process, where the cutoff value C is set to 0. The extracted alignment is the longest chain and the calculated score for the pair is the sum of the scores.

calculated for the remaining regions, and they are evaluated against a defined expected value threshold E_T. The expected value defines the expected amount of random matches with a given score that can be found in a database of a given size. This threshold acts as a filter, so that all found pairs must get high enough calculated expected value E_C, or they are discarded. For example, if a match gets a calculated expected value of 1, one can expect to find 1 random match from the database with the same score as that by pure chance.

It therefore describes the noise level of the database. The value is also correlated with the score S_R, as the expected value decreases exponentially when the score gets higher, i.e. it is more unlikely for a longer match to be found from the background noise. The expected value is calculated from three different factors: length of the query sequence, size of the database and the bit score, which is a normalized version of the raw score obtained in the previous step. Bit score is calculated using the formula:

$$S = \frac{\lambda * R - ln(K)}{ln(2)} \tag{3.3}$$

where λ and K are statistical parameters that can be approximated from the data (Mount, 2001), though in NCBI BLAST they are predefined with the substitution matrix. In the original article, λ was defined as 0.318 and K as 0.13. (McGinnis and Madden, 2004) They are constants and can be thought of as natural scales for the equation. Using the normalized score, the expected value is calculated using the formula:

$$E = mn2^{-S} \tag{3.4}$$

where m is the effective length of the query sequence and n is the effective size of the database. Effective length and size are used rather than the actual length and size. This is due to the fact that no alignments can start at the very end of a sequence, as alignments require some amount of length to not be culled by a threshold. Therefore the whole length of the sequence is not applicable, so an adjusted length is used instead. The effective length of the sequence is the actual length of the sequence minus the expected length of an expanded region. This expected length can be predefined or estimated from the data by calculating a region that would have E_C score of 1 and noting its length. The effective length of the database is then the sum of effective lengths of all sequences in the database. (Mount, 2001)

The expected value threshold is set low, close to zero, as regions with high E_C value could just be background noise. Decreasing the threshold has a side effect, however. Very

short regions, even identical ones, get naturally higher E_C values, as it is impossible for a short region to attain a high score sum. This therefore leads to BLAST being unable of finding shorter regions. Setting the threshold value correctly is thus a compromise between how short regions one wants to find and how many false positives there can be.

Often two or more regions can be combined to make a new, longer aligned region. That is, there is a gap between the two regions. BLAST finds the optimal pair of regions by calculating a new score for all combinations of nearby regions. If there are several options to combine, BLAST uses sum-of-scores method to define which regions should be combined, where the combination with the highest sum of scores minus a calculated gap penalty for the gap is the optimal solution. If no candidates for combinations are found or the scores for combining them are too low, the regions are left as they were.

The final part of the algorithm is to output the gapped Smith-Waterman local alignments of the similar regions between the query sequence and the matched sequences in the database. As explained in the previous section, Smith-Waterman is a local alignment algorithm and thus provides a local alignment between the pairs. The two whole sequences are not aligned against each other, just the regions that match. Example of the output can be seen in Figure 3.9. The example shows how the matches, mismatches and gaps are outputted in the process as well as two possible local alignments, both equally optimal. This output is not always necessary, however. If the only goal is to find sequence offset pairs, i.e. the starting and ending offsets that depict a similar region between two documents, calculating the gapped local alignment is unnecessary and can be skipped.

Examples of the hyperparameters used can be seen in the Table 3.1. Setting these values correctly is vital, as they drastically affect the precision and recall of the algorithm, as explained previously.

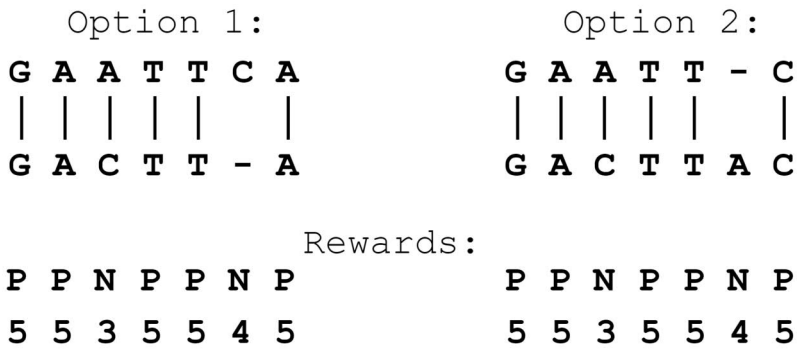

Figure 3.9: Example of the gapped Smith-Waterman local alignment between the sequences ”GAATTCCA” and ”GACTTAC”. The P and N denote whether the reward is positive or negative, respectively. Two possible alignments are shown.

Parameter name	**Identifier**	**Example value**
Seed length	k	4
Seed similarity threshold	T	20
Window size	W	20
Cutoff value	C	0
Accumulated threshold	ST	50
Expected value threshold	E	0.0001
Substitution matrix	MATRIX	BLOSUM62
Gap open penalty	GAPOPEN	3
Gap extend penalty	GAPEXTEND	11

Table 3.1: A table showing the common hyperparameters and an example value for them.

4 Using BLAST to detect text reuse clusters

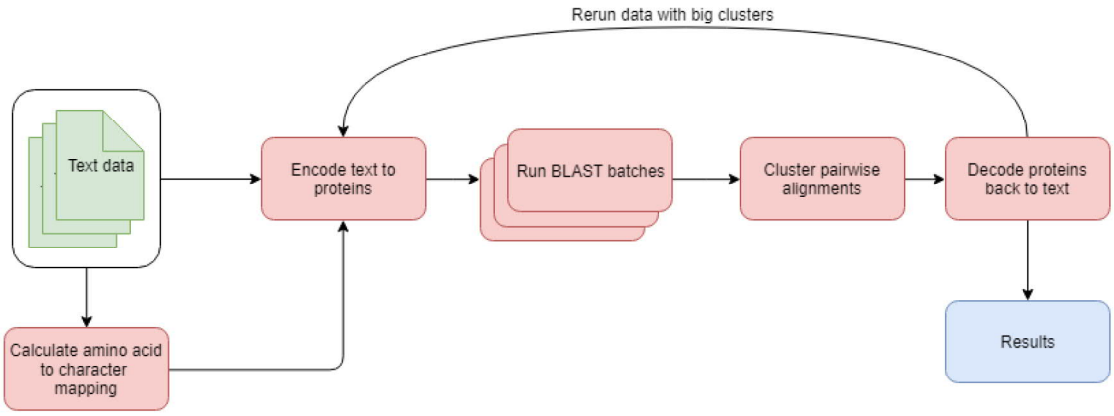

Figure 4.1: Chart showing the full process of running the text version of BLAST.

In this chapter I go through the necessary steps to be able to use BLAST to detect text reuse passages. As an implementation of the BLAST algorithm, I use NCBI BLAST, which I refer to just BLAST in this chapter. I also refer to the aligned and expanded regions as hits. As BLAST is designed for aligning biomedical sequences, it has certain assumptions, such as there only being 23 amino acids. Such assumptions cause issues when trying to extend the applications of the software, due to them being hardcoded into its logic. The biggest issue being the alphabet difference, which is also replicated all around the source code. Therefore rather than rewriting the whole program, I get around this issue by tricking BLAST into thinking that it is dealing with protein sequences, whereas

T h i s i s a n e x a m p l e s e n t e n c e .

==

D S C H C H B E G B N Q F G H G E D G E G

Figure 4.2: Example of the text-to-protein conversion

in reality it is processing natural language.

4.1 Detecting clusters

4.1.1 Data pre-processing

To use BLAST, I must adhere to its assumptions about the data. That is, I can only use amino acid sequences. Therefore I must encode the input data into amino acids. I lowercase the dataset in question, calculate the 23 most used letters and form a simple one-to-one mapping between the letters and arbitrary amino acids. This step also works as a pre-processing step, as it removes all special characters and numbers from the text. I do not include the whitespace character, even though it is naturally the most used character. The whitespace character always appears before and after a word, so it ends up working as a padding, effectively lowering the used word size by two, and simultaneously increasing the database size by 50%. In tests, I also saw that the found hits were more often split into two independent hits if I encoded the whitespace, as well as increasing the needed processing power. Using this mapping I can then encode the whole dataset into proteins, which can then be processed by BLAST. Figure 4.2 shows an example of said conversion.

This method has clear disadvantages, namely the limited alphabet. Sometimes having numerals in the data could be vital, or the data could be using a character system with more letters, such as Chinese, Japanese or Korean, where having just 23 characters might

not be sufficient. A possible fix for these issues would be to create a mapping that is one-to-many instead of one-to-one. That is, map a single symbol into a combination of one or more amino acids. This method would allow arbitrary many symbols to be used in the alphabet. However, this method does come with a caveat. For example, combinations ”AAAAB” and ”AAAAC” are very similar to BLAST, but these could be two completely different symbols in the original data.

Using compound amino acids as symbols would therefore cause two issues. First being doubling the size of the data for every extra character in the combinations. Given a large corpus, this could very easily render the process unusable by today’s computational standards. Second being the possible erroneous caused by the fact that two symbols could share a high percentage of identical amino acids while in reality being completely different.

4.1.2 Clustering

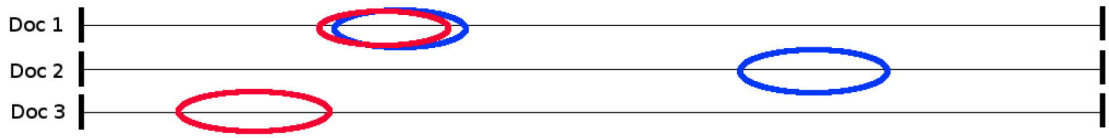

Figure 4.3: Example of the overlap. Two slightly different offset values, yet clearly the same passage. Each color depicts a pair.

Using BLAST, I compare every document against the entire dataset, i.e all pairwise comparisons between the documents. BLAST produces similar pairs, or in my case, text reuse pairs between documents. The pairs contain the document’s ID as well as the starting and ending offset of the sequence where the reuse occurs. BLAST does not automatically cluster the data. If the same text passage is reused in number of documents, each of the identified document pairs may mark a slightly different beginning and ending

offsets due to the matching being fuzzy and because of the massive noise the data may contain.

For instance, if a passage from a document A is reused in documents B and C, the offsets in A will be slightly different between the $A - B$ and $A - C$ pairs. To deal with this problem, I calculate consensus offsets that combine all passages in one document from individual document pairs that are close to each other. In this example, the two passages in A from the $A-B$ and $A-C$ pairs. I define a threshold here, which denotes the minimum similarity to include in the consensus process. By default, the offsets have to be at least 75% overlapping, or the overlapping offsets are left separated. This is another parameter which can be tuned to better fit the data and the purpose of the process. Example of the overlap can be seen in Figure 4.3. Each offset that was combined into a consensus offset was linked to another offset in another document. While calculating the consensus, I map the old offset into the new one. This allows me to then link these offsets together, so that they form a graph. Each of these connected graphs equals a single cluster from the data.

The obtained clusters are still just offset values from encoded sequences. I still need to reverse the encoding process to get the original text back. I do this by taking the original full text and encoding that in reverse, so that I only encode the characters that were previously discarded. As the text is lower cased, I can just encode the characters into an arbitrary uppercase letter without the possibility of collision. I then go through the encoded full text and calculate a mapping from encoded offsets to the original. Using said mapping I can then reverse the process and retrieve the actual text reuse passages.

4.1.3 Sliding window effect

Due to the consensus offsets, it is possible for a ”sliding window” effect to appear. If a particular passage is repeated a lot as well as the neighboring passages in the documents,

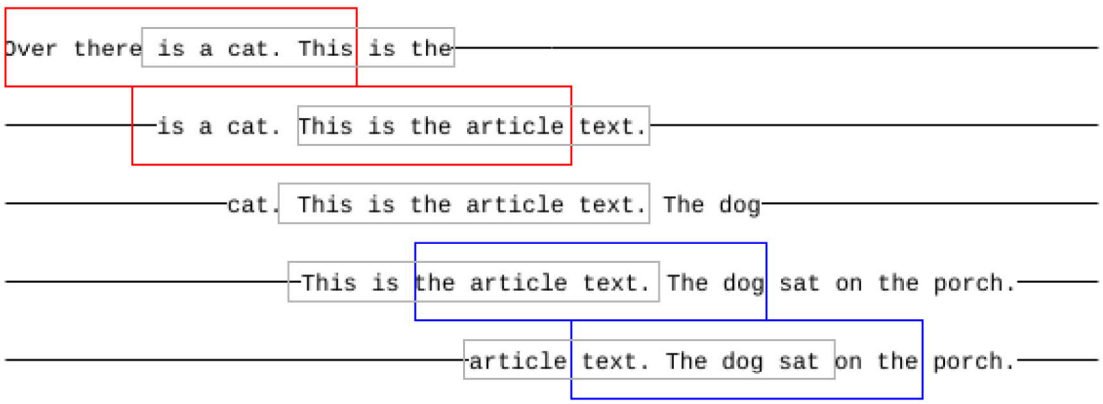

Figure 4.4: Example of the sliding window effect. Red color means the offsets have slided to left, i.e smaller and blue vice versa.

it is possible for the offset values to slide a bit by bit. Example of this can be seen in Figure 4.4. The red color depicts offsets sliding to left, i.e starting and ending offsets are smaller and blue the same but vice versa. Gray shows the actual original text. The overlapping red or blue and gray show from which parts of the text the consensus is calculated. The example is a very extreme case. In real applications such low percentage of total overlap would not be combined.

The middle one with just gray box is the original article and the content that should be separated into a cluster. Here can be seen that slowly previously unseen texts are being added into the cluster. The top and bottom level texts have very little of the original article left in them, and they have nothing in common with each other. In similar fashion, this effect could continue indefinitely, given that the conditions are favorable. This renders the clusters unusable for any statistical use, as they no longer properly reflect the data, as well as make any manual examination very cumbersome.

To fix this issue, I find all clusters that have more than 100 hits and rerun the BLAST process again using just the hits as the dataset. I then go through the pairs and use greedy

search to find documents that have any similarity and add them to a cluster as long as they are not already in a new cluster. This will form clusters that have separate hits in them. Due to the way the sliding window happens, the resulting clusters will share similarities and some hits could possibly fit into two different clusters. As such, this is not a perfect solution, but in the absence of better solutions, it keeps the clusters clean, as none contain multiple different passages.

4.2 Adjusting the algorithm

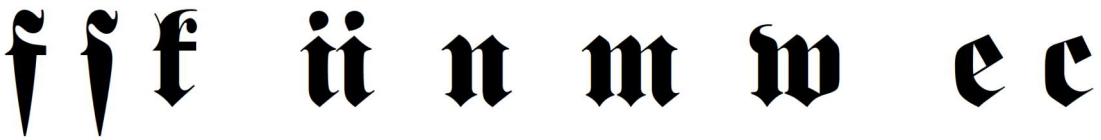

Figure 4.5: Example of similar characters in the Fraktur font. Letters in order: f,s, k,i,i,n,m,w,e,c

The substitution matrix that BLAST uses out-of-the-box is either an identity matrix, or a protein specific matrix, such as BLOSUM62. A protein specific matrix is naturally not a good fit for textual data, as the arbitrarily chosen amino acids in the mapping do not share the same similarities as they do in the real world when they are part of protein sequences. Identity matrix is thus the only choice. Yet due to the nature of the OCR process and fonts themselves, there are several characters that are more likely to be wrongly considered each other. Due to this, using a identity matrix that penalizes all mistakes equally is not the most optimal approach. A better choice is to calculate a new substitution matrix that reflects the errors made by the OCR system, giving BLAST more leeway when it comes to penalizing the OCR noise.

In Figure 4.5 you can see example of similar characters in the Fraktur font. The first

9	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5
-5	**9**	-5	-5	**-4**	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	**9**	-5	**-4**	-5	**-4**	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	**9**	-5	-5	-5	-5	-5	-5
-5	-5	-5	**9**	**-4**	-5	-5	-5	-5	-5	-5	-5	**1**	**-3**	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5
-5	**-4**	-4	**-4**	**9**	-5	**-4**	-5	-5	-5	-5	**-4**	-5	**-4**	-5	-5	-5	**-4**	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	**9**	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-4	-5	**-4**	-5	**9**	-5	-5	-5	-5	**1**	-5	**-4**	-5	-5	-5	-5	-5	-5	**-4**	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	**9**	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5	**9**	-5	-5	-5	-5	-5	-5	-5	-5	**-4**	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5	-5	**9**	-5	-5	-5	-5	-5	-5	-5	-5	**-4**	-5	-4	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	**9**	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	**-4**	-5	**1**	-5	-5	-5	-5	**9**	-5	-5	-5	-5	-5	-5	-5	-5	**-4**	-5	-5	-5	-5
-5	-5	-5	**1**	-5	-5	-5	-5	-5	-5	-5	-5	**9**	**0**	-5	-5	-5	**-4**	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	**-3**	**-4**	-5	**-4**	-5	-5	-5	-5	-5	**0**	**9**	-5	-5	-5	-5	-5	-5	**-4**	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	**9**	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	**9**	-5	-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	**9**	-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	**-4**	-5	-5	-5	-4	-5	-5	-5	**-4**	-5	-5	-5	-5	**9**	-5	-5	**-3**	-5	-5	-5	-5
-5	-5	**9**	-5	-5	-5	-5	-5	**-5**	**-4**	-5	-5	-5	-5	-5	-5	-5	-5	**9**	**9**	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	**9**	**9**	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	**-4**	-5	-5	**-4**	-5	**-4**	-5	**-4**	-5	-5	-5	-3	-5	-5	**9**	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	**9**	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	**9**	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	**9**	-5
-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	**9**

Table 4.1: New substitution matrix. ”Non zero” values are bolded.

two letters, f and s, are nearly identical. The only difference is a small notch that could easily have been faded away in old documents. Other similarities can be seen in m and w, where the only differences are a small notch and one part of m is vertically reversed, or with e and c a small line creating the closed loop in the character e. Characters n and m are also constructed by attaching several times the base of character i and connecting them via small line. Naturally the i character has the superscript dot on top of it, but due the bad quality of the original scans, these parts could easily fade away partly or completely, which causes the OCR system to very easily misrecognize these. As a font, Fraktur is

therefore not the easiest choice for OCR systems.

Kansalliskirjasto has manually annotated and published a dataset of corrected OCR documents. The dataset consists of 470 pages, where they have both the OCR output as well as the manually corrected version (Kettunen and Koistinen, 2018). I encode all word pairs into proteins using the previously calculated protein mapping and align the gold-ocr pairs using Needleman-Wunsch algorithm described in the chapter 3.2. The algorithm finds the global alignment between a gold-ocr word pair. Using these alignments, I can see which characters tend to be replaced by what and how often. Using these alignments I can calculate an error distribution for each character.

I start with the identity matrix used by BLAST, where every value on the diagonal axis is 9 and all other values are -5. This is not the standard identity matrix, where the diagonal values are 1 and the rest are 0. This is due to two things. Firstly BLAST uses this to penalize, so the non-diagonal values have to be negative for the aligning algorithm to ever decrease. Secondly the opening and extending gap values are also used, which would unfairly penalize the system by penalizing a gap more than a mismatch.

Using the error distribution, I scale the values based on their logarithmic frequency. I calculated the ratio of how much a certain letter was used instead of the correct one and scaled the non-diagonal value from -5 all the way up to 9, depending on how high the ratio was. In this configuration most values changed and many were over 0. This, however caused BLAST to produce completely inaccurate results. The substitution matrix had too much positive in it, so in the aligning process random correct characters that appear by chance in two completely different sequences were enough to keep the algorithm running.

Instead of giving the values ability to go all the way to 9, I limited it to 0. This restriction

caused only few values to change, as most values were scaled from -5 to somewhere between -5 and -4.5, which would then be rounded back to -5. I also changed the reward of $W - V$ and $V - W$ to both be 9. This is due to the fact the two letters were interchangeable in Finnish in the 1800s.

Table 4.1 shows the new substitution matrix. Here we can see that while most values stayed at -5, a few got a boost. The small boosts are enough to help BLAST find more results, as demonstrated in the chapter 5.3. An important note about this method is that it is completely domain specific. Other OCR systems or fonts would have different error distributions and interchangeable characters. This matrix thus needs to be recalculated for all the different corpora.

5 Results

5.1 Newspapers and journals, 1771-1910

Multa t\ä@tä fyNlkÄsiii kchtalostu ,ct , Abouil Asi,3 wic!lä ticiun't>t ,mitää>«, »vaalii luiftti iloista M,m<iä Tshiragauissa, ©elä fi:föf3>i'öi että uiUfatfpäim -uhkaisiloui i Hviarat, miinto fu^tiaani 'fatifefi- fuffotai» lÄuja THi roinin, puutarhassa ja, ipici'ilitsi hwi'tt<iiöii fmmiamcrk^iUi ja anoo» »imilyMla,

Mutta tästä synkästä kohtalosta ei Abbul Asib »ielä tiennyt mitään, vaan »ietti iloista elämää TshiraganiSsa. Sekä sis»Stä «ttä ulkoapäin uhkasivat «aarat. mutta sulttaani katseli lukkotaisteluja Tfhiiaaanin puutarhassa ja palkitsi voittajan lunnicnnerleillä ja ar° vonimityksillä.

Figure 5.1: Bad quality hit pair. Both texts are identical in the original scans, but the OCR-version of them are completely different.

In this thesis I focus on a single corpus, the National Library of Finland's (NLF) publicly available collection of newspapers and journals from years 1771 to 1910. The corpus contains mostly Finnish and Swedish documents, but few German and Russian documents are included (Pääkkönen et al., 2016). In total, there are approximately 3 million documents. These old documents have been scanned and using OCR software turned into text. Due to many factors, such as the old age, bad scans, unoptimal OCR program as well as the hard to read font, Fraktur, which used to be prevalent in early modern Finnish, the texts are at times far from accurate. It has been estimated that the average error rate of this corpus is 25-30%. (Kettunen et al., 2016). Example of the bad quality in this corpus is shown in the Figure 5.1.

Running the whole dataset through the previously explained pipeline, I found around 8 million clusters of repeated texts that together have a total of 49 million occurrences longer than 300 characters. Note, however, that some clusters refer to the same, larger repeated news passage, in different lengths. This happens due to the fact that at times the OCR quality is too low, allowing the method to only find fragments of the actual article, thus forming two separate clusters, one with only the few long, intact versions that could be detected and one with all the smaller fragments. Since the surrounding text of a fragment is too dissimilar, it is difficult to establish whether the two clusters could be combined without introducing errors in the data. Therefore the number of clusters or occurrences does not necessarily reflect the actual number of unique text reuse.

In chapter 6, I perform qualitative analysis on the found clusters, where I categorize the clusters into three different overlapping categories.

5.2 Comparison against Passim

As with all such empirical methods, evaluating the actual performance of the system is hard, as there is no gold-standard data against which to compare the results. As such, I compare the accuracy and sensitivity of BLAST against Passim, a popular tool for text reuse detection (Smith et al., 2014) used in many similar studies previously, to see how BLAST works in a relative comparison against the state-of-the-art.

I form a dataset of 2,000 randomly selected documents from the NLF corpus and run both Passim and BLAST with this dataset. For the whole dataset I calculate the coverage using the equation:

$$\text{coverage} = \frac{\text{total amount of unique characters in the clusters}}{\text{total amount of characters in the dataset}} \tag{5.1}$$

System	Coverage
BLAST	0.177
Passim (default)	0.057
Passim (optimized)	0.080

Table 5.1: Text reuse coverage comparison of the BLAST-based method relative to Passim with its settings left at their default values, as well as optimized to maximize recall.

As a metric it signals how big of a fraction of the text in the whole dataset is found to be text reuse. Increasing this number means the software finds more reuse, given that the passages are not false positives. The results are shown in Table 5.1.

The results show that the BLAST based method vastly outperforms Passim in terms of recall. In order to establish that this gain in recall is not at the expense of precision, I sample clusters both randomly and at the very bottom of BLAST similarity scores still acceptable for inclusion in the results and manually verify the proportion of those that are true positives. The proportions are shown in Table 5.2. The results naturally depend on the length of the texts in the cluster, with shorter texts less likely to be correct hits than the longer ones, given a constant alignment score.

To understand to what extent the hits identified as text re-use are dissimilar, I randomly selected 1000 clusters which contain only two hits of at least 300 characters in length. I then calculate the pairwise character alignment between these two hits and measure the proportion of matching characters, i.e. not gaps nor misalignments, using the Needleman-Wunsch algorithm. As shown in Figure 5.2, the alignment values range from around 99% down to as low as 40%, with the bulk of the data in the 70–90% range. For the most part, the repeated texts thus differ in 10–30% of positions, but the difference can be as much as 60%. Partly, these are cases of e.g. advertisements which differ only in numerical values,

Range	BLAST Precison random	BLAST Precison low	BLAST Coverage	Passim Coverage
300 - 350	1.00	1.00	0.108	0.076
250 - 299	1.00	0.94	0.120	0.078
200 - 249	0.94	0.94	0.133	0.079
150 -199	0.92	0.86	0.154	0.080
100 - 149	0.86	0.70	0.177	0.080

Table 5.2: The precision and coverage of the BLAST method on 50 clusters of varying text hit lengths, sampled randomly and at the lowest alignment scores acceptable.

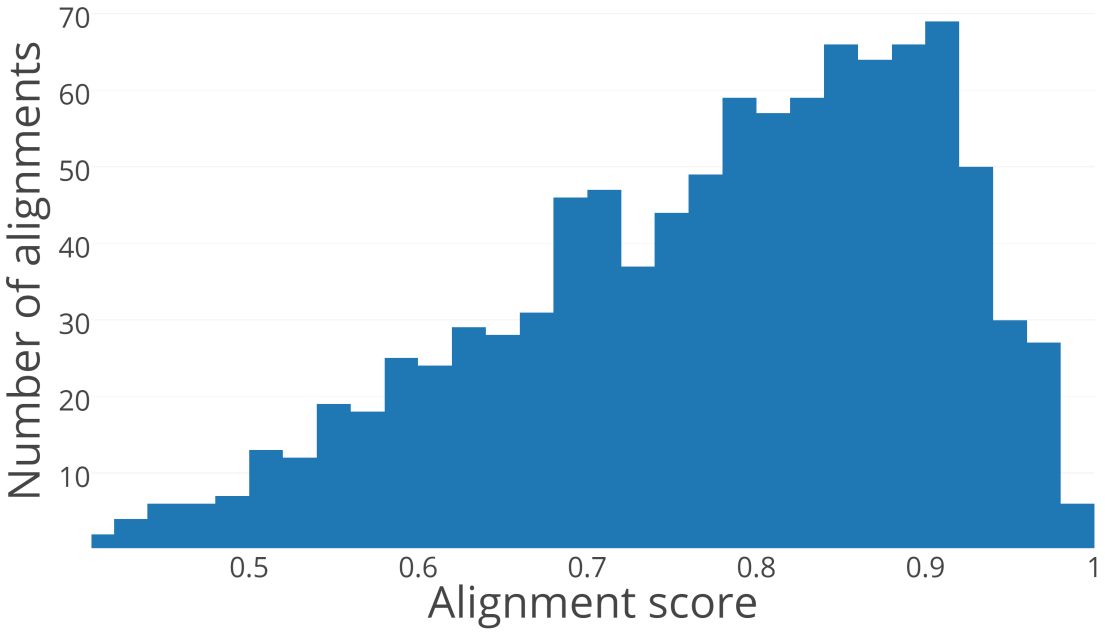

Figure 5.2: The distribution of alignment scores (horizontal axis) and the number of clusters out of 1000 with the given alignment score (vertical axis). Minimum text reuse length is 300.

Substitution Matrix	Coverage
Identity	0.079
Data specific	0.069

Table 5.3: Results of learning a better substitution matrix over the identity matrix.

but partly these are in fact fully identical texts with a massive OCR error rate. It is due to these hits where the difference can be as vast as 60% that I cannot use software such as Passim, but must use a more heavy solution.

The gain in recall comes at the expense of compute time, with BLAST being about three orders of magnitude slower than Passim. Applying BLAST to the entire NLF dataset required around 150,000 CPU-core hours. This is certainly out of reach for a single computer, but well within modern cluster computing resources, especially since the historical text collection is static and the run only needs to be carried out once.

5.3 Custom substitution matrix

To test the new substitution matrix, I created a small dataset of 1000 random documents from the NLF dataset and ran it through the full pipeline twice, one with the normal identity substitution matrix and one with the custom one. I evaluate the results using the same coverage measure explained in the previous section. I manually read through the results to confirm that the results are not erroneous and are in fact true positives. As seen in table 5.3, using the new substitution matrix the coverage increases by 0.01 p.p, a 15% relative increase in coverage. In a small dataset such as this test set, the increase is not huge, but a 15% relative increase in a dataset of million documents is already substantial.

6 Qualitative analysis

Newspaper reprinting and reusing can be classified into three different, though sometimes overlapping, categories: every day text reuse, long-term reuse and viral news. A large majority of text reuse is merely reprints of advertisements or announcements that were reused just a couple of times. Sometimes they were reprinted in a short period, sometimes after a year. Some, however, also contain long-term reuse. That is, there is a significant gap between the dates. Finally there are also reused passages that achieved viralness.

In this chapter I further analyze the clusters I got from applying the BLAST text reuse detection algorithm to the entire NLF dataset. To this end, I categorize them into the three different categories. I analyze the difference of these categories and provide examples of what the categories are or what they can be used for.

6.1 Advertisements and announcements

The clusters from the first category are the ones that are not especially distinctive on their own. These include advertisements and announcements, poems and stories as well as normal, every day news that do not achieve viralness. These clusters make up the large majority of the found clusters. A product being sold and its prices, or timetables for the post office, for example. These clusters are independently less interesting than actual news, but when analyzed en masse, they show how reuse occurred between different cities and newspaper titles. They therefore show the bigger picture of text reuse in the 1800s.

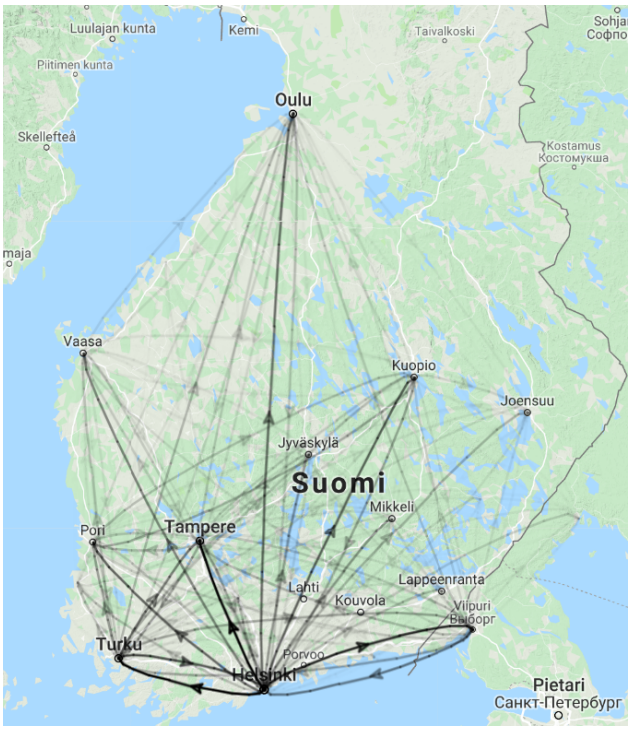

Figure 6.1: A map of Finland, where the amount of text reuse between two cities can be seen.

This group overlaps with the other two groups, as sometimes certain advertisements or announcements were viral or contained long term reuse.

Figure 6.1 shows a map of Finland, where I calculated from which cities reuses usually originated. In this example I focused on the clusters from the years 1880 to 1890. Here I looked at the clusters one at a time, took the first occurrence of a reused passage and used that as the origin location and used the rest as the destinations. The darker the line is, the more text reuse took place between the cities in the given direction. From the figure can be seen that the most relevant cities for communication are the main hubs, Turku and Helsinki, but also some of the smaller ones, like Sortavala and Viipuri. Interestingly some of the smaller cities shared reuses between each other as well, bypassing the main communication hubs. These are most likely smaller scale advertisements that had no need to appear in the southern cities.

6.2 Long term text reuse

Due to the large time frame of the NLF dataset, clusters considered as long term reuse can be found from the data. That is, they were either reprinted regularly for many years or contained a long gap between two reprints. These clusters are the ones that are extremely hard to find manually, as reading and remembering articles that have over a hundred years between them is nearly impossible for a human. It therefore opens up new research questions and answers, e.g. what kind of news were reprinted after a century and why.

Example of this can be seen in Table 6.1, where the reprint dates of a patriotic song can be seen. It was first published once in 1821 and then three times after a gap of 70 years. These kinds of clusters are not unheard of in the data, where a passage has been reprinted after many years for different reasons, such as commemorating an event that took place in the past, showing a slice of the past by reprinting what was printed several decades before or even for political gain.

Cluster	Date	Title
639828	1821-03-03	Åbo Morgonblad
639828	1891-02-20	Nya Pressen
639828	1891-02-20	Folkwännen
639828	1891-02-21	Åbo Tidning

Table 6.1: Reprints of a patriotic song.

6.3 Viral news

The third group of text reuse are the viral news. According to a rough manual overview, their amount increases rapidly after the Crimean War took place. (Vesanto et al., 2017a). These clusters that achieved viralness are the ones that contained information that had to

Place	Date	Title
Helsinki	1906-11-07	Uusmaalainen
Helsinki	1906-11-07	Helsingin Sanomat
Turku	1906-11-08	Uusi Aura
Helsinki	1906-11-08	Elämä
Tampere	1906-11-08	Tampereen Sanomat
Turku	1906-11-08	Sosialisti
Helsinki	1906-11-08	Uusi Suometar
Jyväskylä	1906-11-09	Suomalainen
Oulu	1906-11-09	Kaleva
Kuopio	1906-11-09	Pohjois-Savo
Tampere	1906-11-09	Kansan Lehti
Viipuri	1906-11-09	Karjala
Sortavala	1906-11-10	Laatokka
Heinola	1906-11-10	Heinolan Sanomat
Savonlinna	1906-11-10	Keski-Savo
Joensuu	1906-11-10	Karjalatar
Lahti	1906-11-11	Lahden Lehti
Kemi	1906-11-12	Pohjois-Suomi
Kristiina	1906-11-12	Etelä-Pohjanmaa
Lahti	1906-11-13	Lahti

Table 6.2: Reprints of a bank robbery news.

be quickly spread all along the country. They are therefore among the most interesting clusters to observe from a scientific point of view.

Example of a viral news can be seen in Table 6.2. The article was printed 20 times, all within a single week. The article was first published in Helsinki and the next day in the two other main cities, Turku and Tampere. Then, in the span of 5 days, it circulated to smaller and smaller cities.

The three categories of text reuse could also overlap. Long term reuse, for example, might later on transform into viral texts. Commemorations for an event that took place a century ago might be interesting to reprint in several papers, for example. Advertise-

ments, while not actually spreading naturally like news due to their nature, can still seem viral. High-end advertisements could be paid to be printed in many cities and titles within a short time frame.

To further examine the most viral clusters, I need to automatically find them. Seeing as many advertisements were quickly reprinted in many papers, mere reprint speed nor the amount are sufficient enough to be able to tell whether an article is viral or not. To that end, I calculate a virality measure for all of the clusters by focusing on three different criteria from the cluster: the amount of different geographic locations, amount of different newspaper titles as well as the time it took for the news to circulate. Therefore, the equation used for the virality measure calculation is:

$$\text{score} = \frac{\text{unique locations}}{\text{total unique locations}} * \frac{\text{unique titles}}{\text{total unique titles}} * \frac{1}{\text{number of elapsed days}} * 100 \quad (6.1)$$

The equation thus ranks the cluster with a value between 0 and 100, where it penalizes the score heavily if even one of the three components is too small. The motivator behind this metric is that a viral piece of news should be spread all around the country, it should appear in many different titles even inside the same city and it should circulate relatively quickly. However, it is not unheard of for there to be an article that first went viral and was reprinted in many locations to also be reprinted several days, even years, later in another paper. The last reprint would then heavily penalize the score, thus skewing the results. To remedy this, I look at all the reprints in the cluster and ignore in the scoring phase any reprints that are clearly statistical outliers, using Tukey fences as the threshold.

The vast majority of the clusters, up to almost 97%, receive a below 1 viral score, i.e

they are the opposite of a viral news. Naturally a lot of them are not even news, but instead they can be advertisements, announcements, poems etc. However some of them are actual news, but for a number of different reasons they did not gain the traction to become viral. This could have been caused by many different factors, such as location, importance, censorship at the time etc.

If the obviously non-viral clusters with score being smaller than one are ignored, there is still a significant amount of real, viral clusters, even though the vast majority of the clusters were not. This can be seen in the Figure 6.2. In total 470,896 clusters had viral score bigger than 1.

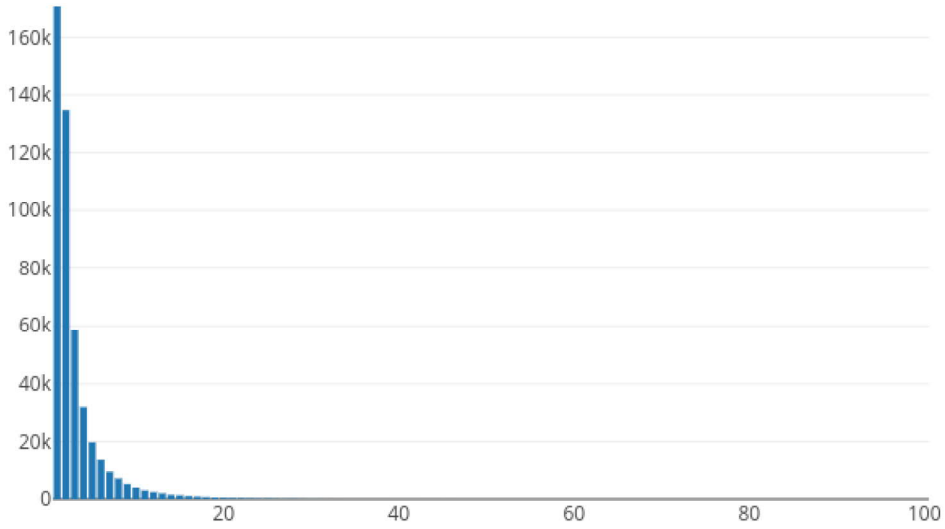

Figure 6.2: The viralness score distribution of all clusters.

While a viral score of just over one is still relatively small and might still not be con-

sidered as viral, clusters with significantly higher score can safely be considered as one. For example, in the results, clusters with score just above 1 could achieve it by spreading to 5 different unique locations and titles in a day, or to 8 different locations and to 10 different titles in 5 days. It can naturally be achieved with different combinations, but in all of them the spreading is fairly tame. However, as one possible combination, a cluster with viral score 50 already had to spread to 29 unique locations and 62 titles in 2 days. This is clearly a sign of a viral news and not just an accident. From the whole dataset the algorithm only found 81 clusters with a higher than 50 virality score. Setting a threshold for viralness is thus a compromise between how many results one wants to find and how viral the clusters have to be.

There exist clusters that receive a high virality score, where they appear everywhere simultaneously, having a time span of one day. One could argue that these might not actually be viral clusters, but instead are coordinated messages that had to be shown everywhere in the country, such as an important event or even a high-end advertisement. On the other hand, these could be real, viral news, where other issues caused the low time span, such as the algorithm missing the original passage that was printed earlier, or if the original article did not even originate from newspapers.

As it stands, the viral score works well enough, though it is not a perfect calculation, as it has some naive assumptions. For example, merely using the inverse of the elapsed time forms a linear function. This means that a difference of 10 days means 10 times worse viral score. This, however, does not accurately reflect the truth. In real life, especially in the 18th century, an article being reprinted again after a week could very well be just as viral as one being reprinted in 4 days. Therefore using a linear function might not properly reflect the reality. Using a different function, such as exponential decay in the time span, could be more accurate. This would not penalize small differences in the

beginning phase nearly as much.

7 Conclusion

In this thesis I propose a novel method of detecting text reuse from a noisy data, such as OCR read (Optical Character Recognition) historical newspapers. The proposed method achieves its new, state-of-the-art precision and recall by leveraging BLAST, an algorithm designed for comparing and aligning biomedical sequences. I explain the BLAST algorithm in depth by going through the algorithm step-by-step. I also explain two other alignment methods that are used in this thesis. I propose a novel method of leveraging BLAST to align natural language and show the different necessary steps for it to work. I also demonstrate the caveats of this and provide a workaround for them.

I demonstrate the effectiveness of the text reuse detection algorithm by applying the method to a dataset of over 3 million OCR read historical newspapers. I also compare it to the previous state-of-the-art method and show that BLAST outperforms it by a massive margin. I verify the results of the tests by manually reading through them to make sure they are not false positives.

I analyze the resulting clusters from applying the method to the dataset and categorize them into three categories: every day reuse, long term reuse and viral news. I explain the difference of these and provide examples of them. I also demonstrate a novel way of calculating a virality measure for the clusters, which can then be used to rank the clusters for observation.

There are several issues that still require further work. The BLAST algorithm is a just fit, but the readily available implementations of it make assumptions about the data. An implementation of the algorithm that does not assume that the data consists of biomedical sequences would be helpful, and for certain datasets, mandatory. As it stands, the method is limited to 23 characters, which is not nearly enough for certain languages with a large alphabet. Implementing a version that supports arbitrary alphabet would then allow the whole text encoding process to be skipped.

The viral score measure works, but it could still be developed further. As it is, it does not fully reflect the realistic viralness due to the linearity of the time span penalty. The time span should be penalized in a non-linear fashion, by using a exponential decay, for example.

In this thesis I categorize the clusters based on the cluster meta data, i.e. how fast they spread and where. However, it would be helpful to classify the clusters based on the content rather than the meta data. This would allow easier observation of just news, for example. This, however, is not a straightforward task, as sometimes the content of the cluster is short in length and may contain a lot of OCR noise. These together make it hard for even a human to detect whether the content is advertisement, announcement, poem or news.

In conclusion, bridging two different fields, in this thesis bioinformatics and natural language processing, is clearly a viable solution to developing more robust software that can beat the previous state-of-the-art. Having a method designed for another purpose does not exclude it from being used for different purposes, as shown by the protein encoding process of the BLAST text reuse algorithm. The fully open-sourced code for running the

proposed method is documented and available at `https://github.com/avjves/textreuse-blast`.

References

S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

D. Bamman, M. Carney, J. Gillick, C. Hennesy, and V. Sridhar. Estimating the date of first publication in a large-scale digital library. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, volume 00, pages 1–10, June 2017. doi: 10.1109/JCDL.2017.7991569.

T. Batane. Turning to turnitin to fight plagiarism among university students. 13:1–12, 04 2010.

M. Bendersky and W. B. Croft. Finding text reuse on the web. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 262–271, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-390-7. doi: 10.1145/1498759.1498835.

D. T. Citron and P. Ginsparg. Patterns of text reuse in a scientific corpus. *Proceedings of the National Academy of Sciences*, 112(1):25–30, 2015. ISSN 0027-8424. doi: 10.1073/pnas.1415135111.

P. Clough, R. Gaizauskas, S. S. L. Piao, and Y. Wilks. Meter: Measuring text reuse. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 152–159, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073110.

W. Commons. File:needleman-wunsch pairwise sequence alignment.png — wikimedia commons, the free media repository, 2018a. URL https://commons.wikimedia.org/w/index.php?title=File:Needleman-Wunsch_pairwise_sequence_alignment.png&oldid=317728547. [Online; accessed 15-December-2018].

W. Commons. File:smith-waterman-algorithm-example-step2.png — wikimedia commons, the free media repository, 2018b. URL https://upload.wikimedia.org/wikipedia/commons/2/28/Smith-Waterman-Algorithm-Example-Step2.png. [Online; accessed 15-December-2018].

W. Commons. File:smith-waterman-algorithm-example-step3.png — wikimedia commons, the free media repository, 2018c. URL https://upload.wikimedia.org/wikipedia/commons/e/e6/Smith-Waterman-Algorithm-Example-Step3.png. [Online; accessed 15-December-2018].

S. R. Eddy. Where did the blosum62 alignment score matrix come from? *Nature Biotechnology*, 22:1035–1036, 2004.

T. Elsayed, J. Lin, and D. W. Oard. Pairwise document similarity in large collections with mapreduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, HLT-Short '08, pages 265–268, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

R. Gaizauskas, J. Foster, Y. Wilks, J. Arundel, P. Clough, and S. Piao. The meter corpus: A corpus for analysing journalistic text reuse. In *Proceedings of the Corpus Linguistics 2001 Conference*, 01 2001.

P. Gupta and P. Rosso. Text reuse with acl: (upward) trends. In *Proceedings of the ACL-2012 Special Workshop on Rediscovering 50 Years of Discoveries*, ACL '12, pages 76–82, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

P. A. V. Hall and G. R. Dowling. Approximate string matching. *ACM Comput. Surv.*, 12 (4):381–402, Dec. 1980. ISSN 0360-0300. doi: 10.1145/356827.356830.

A. Huang. Similarity measures for text document clustering. pages 49–56, 2008.

K. Kettunen and M. Koistinen. Re-ocr in action-using tesseract to re-ocr finnish fraktur from 19 th and early 20 th century newspapers and journals. 04 2018.

K. Kettunen, T. Pääkkönen, and M. Koistinen. Between diachrony and synchrony: Evaluation of lexical quality of a digitized historical finnish newspaper and journal collection with morphological analyzers. In *Baltic HLT*, 2016.

J. Lee. A Computational Model of Text Reuse in Ancient Literary Texts. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 472–479, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

S. McGinnis and T. L. Madden. Blast: at the core of a powerful and diverse set of sequence analysis tools. *Nucleic Acids Res*, 32:20–25, 2004.

D. Metzler, Y. Bernstein, W. B. Croft, A. Moffat, and J. Zobel. Similarity measures for tracking information flow. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pages 517–524, New York, NY, USA, 2005. ACM. ISBN 1-59593-140-6. doi: 10.1145/1099554.1099695.

D. W. Mount. Bioinformatics–sequence and genome analysis. *CSHL, New York*, pages 75–85, 2001.

S. B. Needleman and C. D. Wunsch. A general method applicable to the search for

similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

T. Pääkkönen, J. Kervinen, A. Nivala, K. Kettunen, and E. Mäkelä. Exporting Finnish Digitized Historical Newspaper Contents for Offline Use. *D-Lib Magazine*, 22(7), 2016.

M. Potthast, B. Stein, A. Barrón-Cedeño, and P. Rosso. An evaluation framework for plagiarism detection. 2:997–1005, 01 2010.

G. Salton and M. Lesk. Computer evaluation of indexing and text processing. *Journal of the ACM (JACM)*, 15(1):8–36, 1968. doi: 10.1145/321439.321441.

M. A. Sánchez-Pérez, G. Sidorov, and A. F. Gelbukh. A winning approach to text alignment for text reuse detection at pan 2014. In *CLEF*, 2014.

D. Smith, R. Cordell, and E. Maddock Dillon. Infectious texts: Modeling text reuse in nineteenth-century newspapers. In *2013 IEEE International Conference on Big Data*, pages 86–94, Oct 2013. doi: 10.1109/BigData.2013.6691675.

D. A. Smith, R. Cordell, E. M. Dillon, N. Stramp, and J. Wilkerson. Detecting and modeling local text reuse. In *Proceedings of the 14th ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 183–192. IEEE Press, 2014.

D. A. Smith, R. Cordell, and A. Mullen. Computational methods for uncovering reprinted texts in antebellum newspapers. *American Literary History*, 27(3):E1–E15, 2015.

A. Vesanto, A. Nivala, H. Rantala, T. Salakoski, H. Salmi, and F. Ginter. Applying blast to text reuse detection in finnish newspapers and journals, 1771-1910. In *Proceedings of the NoDaLiDa 2017 Workshop on Processing Historical Language*, pages 54–58, 2017a.

A. Vesanto, A. Nivala, T. Salakoski, H. Salmi, and F. Ginter. A system for identifying and exploring text repetition in large historical document corpora. In *Proceedings of the 21st Nordic Conference on Computational Linguistics, NoDaLiDa, 22-24 May 2017, Gothenburg, Sweden*, number 131, pages 330–333. Linköping University Electronic Press, 2017b.

M. Waterman, T. Smith, and W. Beyer. Some biological sequence metrics. 1976.

Appendix A A System for Identifying and Exploring Text Repetition in Large Historical Document Corpora

A System for Identifying and Exploring Text Repetition in Large Historical Document Corpora

Aleksi Vesanto,[1] **Asko Nivala,**[2,3] **Tapio Salakoski,**[1] **Hannu Salmi,**[2] **and Filip Ginter**[1]
[1]Turku NLP Group, Department of FT
[2]Cultural History
[3]Turku Institute for Advanced Studies
University of Turku, Finland
first.last@utu.fi

Abstract

We present a software for retrieving and exploring duplicated text passages in low quality OCR historical text corpora. The system combines NCBI BLAST, a software created for comparing and aligning biological sequences, with the Solr search and indexing engine, providing a web interface to easily query and browse the clusters of duplicated texts. We demonstrate the system on a corpus of scanned and OCR-recognized Finnish newspapers and journals from years 1771 to 1910.

1 Introduction

The task of finding repeated passages from old newspapers and magazines is relevant to the historians who study the spread of news in time and space. The underlying corpora – in our case scanned and OCR-transcribed newspapers and journals, some over 200 years old – pose a number of technical challenges. Firstly, the size of the corpora is large, in the millions of pages range. And, more importantly, the text produced using OCR is often of poor quality – sometimes nearly unreadable as shown in Figures 1 and 2. This makes the corpora inaccessible to commonly used fast methods such as Passim (Smith et al., 2014) which rely on identifying seed overlaps that are several full words in length, a rare occurrence in our data whose error rate has been estimated to 25-30% in terms of words, depending on period of print (Kettunen et al., 2016).

In this demo, we present a system for identifying text repetitions and forming their clusters using BLAST (Altschul et al., 1990), a software developed to compare and align biological sequences. To browse and search these clusters, we index them using Solr, an open-source search engine, and provide a web interface that is capable of searching and visualizing these repeated text clusters and their associated metadata.

We demonstrate the software and its web interface on a corpus of OCR scanned old Finnish newspapers and journals from years 1771 to 1910, around 3 million pages in total.

2 Software Architecture

2.1 Data Preprocessing and Indexing

NCBI BLAST is built for fuzzy-aligning protein and nucleotide sequences and querying massive sequence databases. As such, it seems an ideal tool for the task, but the assumption of working with biological data is ubiquitous throughout the BLAST codebase, and it cannot be easily used for matching arbitrary alphabets. Therefore, to apply BLAST in our setting, we need to first encode our whole corpus into protein sequences composed of an alphabet of 23 amino acids. As we are limited by the number of distinct amino acids, we can only map the 23 most common lowercase letters in our corpus to distinct amino acids. We then lowercase our corpus and replace all characters using this mapping. Characters that do not have an equivalent in our mapping are discarded – and naturally restored later. This encoding also simultaneously works as a preprocessing method, as the documents have a lot of noise in them in the form of arbitrary characters and spaces. These characters are not among the 23 most common letters, so they are discarded in the encoding process. Interestingly, although space is the most used character in the corpus, we found that discarding spaces nevertheless makes the BLAST query process more than twice as fast and the hits we find are also slightly longer. Once encoded into protein sequences, the documents are indexed using BLAST for a subsequent fast retrieval.

Proceedings of the 21st Nordic Conference of Computational Linguistics, pages 330–333,
Gothenburg, Sweden, 23-24 May 2017.

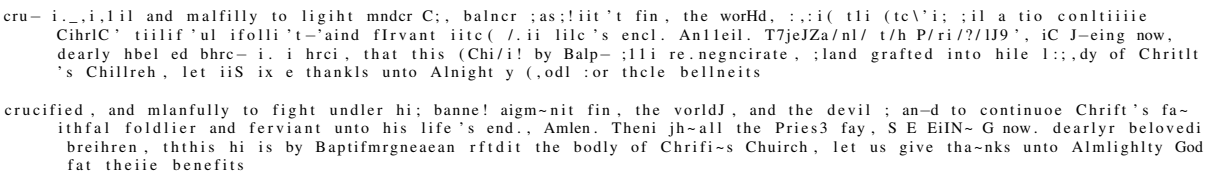

Figure 1: A hit pair from a run with ECCO dataset. (OCR-scanned books from 18th century)

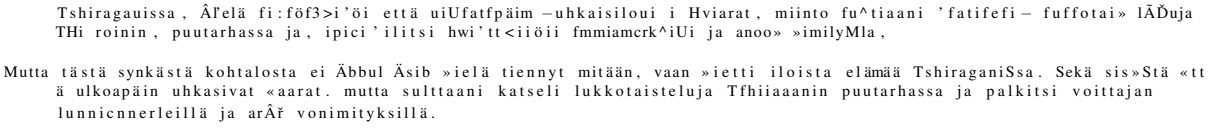

Figure 2: A hit pair from Finnish newspapers.

2.2 Clustering

Every document in the corpus, 3 million pages in our case, is subsequently matched by BLAST against the entire indexed collection – i.e. all pairwise comparisons are calculated. The matching document pairs contain the starting and ending offsets from each document, which we use to connect and combine pairs that share a text passage with a sufficient overlap. Because the matching is fuzzy and the texts are very noisy, if the same text passage is reused in a number of documents, each of the identified document pairs will mark a slightly different beginning and end of the passage. For instance, if a passage from a document A is reused in documents B and C, the offsets in A will be slightly different between the A-B and A-C pairs. To deal with the problem, we calculate consensus indexes that combine all passages in one document from individual document pairs that are close to each other – in our example the two passages in A from the A-B and A-C pairs. We do this by averaging the starting and ending indexes of passages that overlap by at least 80%, obtaining consensus passages.

After identifying all the distinct consensus passages for each document, we create a graph with the consensus passages in individual documents as nodes, and edges between corresponding passage pairs. Subsequently, we extract all connected components from the graph, providing us with an initial estimate of clusters of documents that share a passage. The identification of passage clusters through connected components in the graph can be seen as a high recall method. A stray edge – not uncommon in the noisy data – may connect two otherwise disjoint clusters together. To deal with this, we separate these clusters using community detection. To this end, we apply the Louvain method (Blondel et al., 2008) which identifies communities within the connected components of the graph and we subdivide those connected components that have several distinct, highly-connected communities (subcomponents). This removes the likely stray edges that were connecting them. After this subdivision, we obtain the final clusters and the nodes within them are the repeated text passages we seek.

2.3 Finnish newspapers

We applied our system to old OCR-scanned Finnish newspapers and journals from years 1771 to 1910, around 3 million pages in total. We found nearly 8 million passage clusters containing around 49 million repeating passages. We only considered hits that are 300 characters or longer, as the shorter hits would either be too fractioned to be useful or they are just boilerplate text. The most computationally intensive part of the process is running the BLAST queries, which took 150,000 CPU-core hours. Clearly, a dataset of this size requires an access to a cluster computer, which is not surprising given the complexity involved in fuzzy-matching 3 million pages of text against each other. This computationally intensive step however only needs to be performed once and its results can be reused at a later point.

3 Web User Interface

For the user interface, we index our data with Solr. More specifically, we index the data as nested doc-

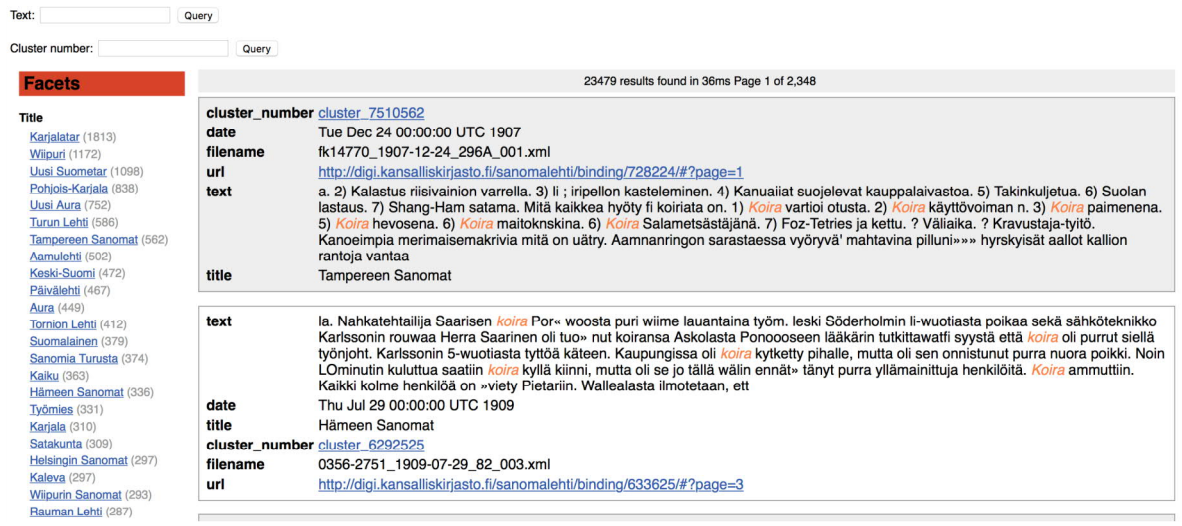

Figure 3: A screenshot showing the user interface.

uments, where the parent document is the cluster and child documents are the hits within that cluster. Solr is capable of querying the data very efficiently, easily allowing for a swift, real-time search. Solr has built-in support for Apache Velocity Template Engine and out of the box it provides a simple browse page where one can browse the query results. Using this template engine, we implement an easy-to-use interface suitable to the nature of the data.

A screenshot of a result page is shown in Figure 3. At the top, a search field allows a free text search. Below is a field for direct search by cluster number. This will result in all hits that belong to that cluster as well as other information about the cluster, such as average length of hits, number of hits and the year of its first occurrence. On the right, we see a small snippet of the results. For every matching hit we can see the name of the original file, date when that issue was published, the name of the newspaper or journal, URL for viewing the original scanned document online, cluster number and the text itself with the query hits highlighted. Clicking the cluster number link shows all hits, i.e. occurrences of the same repeated text passage, within the cluster. Finally on the left we have a facet view, currently giving an overview of hits from a specific magazine. The rich query language employed by Solr gives us the capability of performing fuzzy and proximity search, which is especially useful in our case of low-quality OCR-recognized documents.

As one would expect from a mature search engine like Solr, querying this large collection of repeated text clusters is effortless and real-time. For instance, querying for *kissa*, the Finnish word for *a cat*, found over 23,000 results, returning the first page of 10 results in 38ms.

4 Conclusions

The ability to identify text repetition in large historical corpora is of great importance to historians, and the problem of fuzzy match in large text collections is of a broader interest in corpus research and NLP in general. We have presented a fully implemented and currently deployed software and web interface to identify repeated text passages in large corpora of poor OCR quality, and present them through a simple web interface. We have shown that the BLAST algorithm works efficiently in identifying regions of similarity in historical text corpora, even in cases where the quality of OCR is extremely low, for instance where the original text has been printed with Gothic typeset (*Fraktur*), or with poor paper and ink quality. The development of new tools for text reuse detection is essential for further enhancement of the use of scanned historical documents, and work with noisy corpora in general.

Acknowledgments

The work was supported by the research consortium *Computational History and the Transformation of Public Discourse in Finland, 1640-1910*, funded by the Academy of Finland. Computational resources were provided by CSC — IT Centre for Science, Espoo, Finland.

References

Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. 1990. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, Oct.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008.

Kimmo Kettunen, Tuula Pääkkönen, and Mika Koistinen. 2016. Between diachrony and synchrony: Evaluation of lexical quality of a digitized historical finnish newspaper and journal collection with morphological analyzers. In *Baltic HLT*.

David A. Smith, Ryan Cordell, Elizabeth Maddock Dillon, Nick Stramp, and John Wilkerson. 2014. Detecting and modeling local text reuse. In *Proceedings of the 14th ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '14, pages 183–192, Piscataway, NJ, USA. IEEE Press.

Appendix B Applying BLAST to Text Reuse Detection in Finnish Newspapers and Journals, 1771–1910

Applying BLAST to Text Reuse Detection in Finnish Newspapers and Journals, 1771–1910

Aleksi Vesanto
Turku NLP Group
Department of FT
University of Turku
aleksi.vesanto@utu.fi

Asko Nivala
Cultural History and
Turku Institute for Advanced Studies
University of Turku
asko.nivala@utu.fi

Heli Rantala
Cultural History
University of Turku
heli.rantala@utu.fi

Tapio Salakoski
Turku NLP Group
Department of FT
University of Turku
tapio.salakoski@utu.fi

Hannu Salmi
Cultural History
University of Turku
hannu.salmi@utu.fi

Filip Ginter
Turku NLP Group
Department of FT
University of Turku
filip.ginter@utu.fi

Abstract

We present the results of text reuse detection, based on the corpus of scanned and OCR-recognized Finnish newspapers and journals from 1771 to 1910. Our study draws on BLAST, a software created for comparing and aligning biological sequences. We show different types of text reuse in this corpus, and also present a comparison to the software Passim, developed at the Northeastern University in Boston, for text reuse detection.

1 Introduction

The dataset of the National Library of Finland (NLF) contains 1.95 million pages of digitized historical newspapers and journals from 1771 to 1910. Approximately half of the content is in Swedish, the other half in Finnish, although there are also a few German and Russian papers included (Pääkkönen et al., 2016). We aim to trace the influential texts that were copied and recirculated in Finnish newspapers and journals in this time period. This is done by clustering the 1771–1910 NLF corpus with a text reuse detection algorithm. Our approach enables us to study the dissemination of news and other information and to reconstruct the development of the newspaper network as a part of Finnish public discourse: What kinds of texts were widely shared? How fast did they spread and what were the most important nodes in the Finnish media network?

Our research project builds on a similar study of nineteenth-century US newspapers by Ryan Cordell, David A. Smith and their research group (Cordell, 2015; Smith et al., 2015). However, in contrast to the US press, the nineteenth- and early twentieth-century Finnish newspapers were typically printed in the *Fraktur* typeface, which (together with other possible sources of noise) poses unusual difficulties for Optical Character Recognition (Kettunen, 2016). To solve this problem, we have developed a novel text reuse detection solution based on BLAST (Vesanto et al., 2017) that is accurate and resistant to OCR mistakes and other noise, making the text circulation and virality of newspaper publicity in Finland a feasible research question.

2 Detecting Text Reuse

In the nineteenth century, contemporaries saw newspapers as reflections of modern culture. Many phenomena were amplified by the increasing power of the press, including urbanization, consumerism, and business life. The changes in transport technology led to more efficient distribution of information. Before 1880s, there was no copyright agreement to regulate the free copying of texts, which became a distinctive feature of the press. To understand this process, it is essential to analyze how texts were copied and reprinted.

In Finland, newspaper publishing started slowly, the first paper being *Tidningar Utgifne af et Sällskap i Åbo* in 1771. According to the NLF metadata, in 1850 there were only ten papers and six journals. A rapid upheaval occurred at the

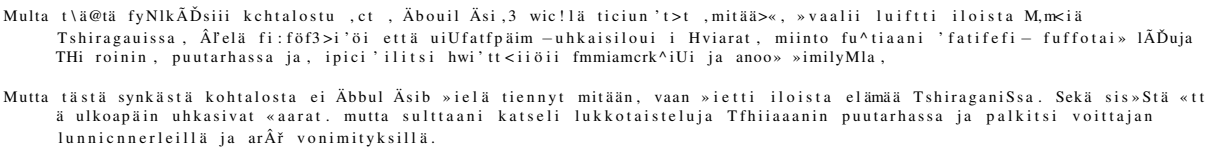

Figure 1: Example of how low the OCR quality can be. Both passages are identical in the original issues.

end of the century, resulting in 89 papers and 203 journals in 1900. The volume of the press was thus very limited during the first half of the century, which also means that text reuse was small-scale. Towards the end of the period the situation changed dramatically, offering more volume for viral chains of reprints. These chains had different origins: they were internal chains within the press, translations from abroad, stories from books, telegrams, or official announcements. Therefore, text reuse detection can shed essential light on how information flowed between centers within the country and how, in the end, Finnish press participated in the global circulation of information.

The primary obstacle in detecting text reuse in the NLF dataset is the poor OCR recognition rate, as illustrated in Figure 1. This makes any approach which assumes exact seed overlaps of several words in length infeasible, and calls for a fuzzy matching method highly tolerant to noise. To this end, we have applied BLAST (Altschul et al., 1990), a sequence alignment software developed for fast matching of biological sequences against very large sequence databases. The main features of BLAST are speed and the ability to retrieve also distantly related sequences – which in our case translates to the ability to withstand the OCR noise present in the data. We index each page of the NLF data as a sequence in BLAST, translating the 23 most common lowercase letters into the amino-acid sequences which BLAST is hard-coded to handle, and subsequently matching the pages in an all-against-all scenario, and post-processing the results to recover the repeated text segments. We choose not to describe the technical details of the process in this paper, and rather focus on the results obtained. The implementation will be made available as open-source software and in the following, we focus on presenting the main results in context of processing historical texts.

System	% of text
BLAST	0.177
Passim (default)	0.057
Passim (optimized)	0.080

Table 1: Text reuse recall comparison of the BLAST-based method relative to Passim with its settings left at their default values, as well as optimized to maximize recall.

3 Text Reuse Clusters – Quantitative Analysis

In total, we found around 8 million clusters of repeated texts that have a total of 49 million occurrences (hits) longer than 300 characters. Note, however, that some clusters refer to the same, larger repeated news piece, in different lengths. This is due to the fact that at times the OCR quality is too low, allowing only for a shorter hit to be identified in some of the repetitions of an otherwise larger text. Since the surrounding text of a shorter hit is too dissimilar (which, after all, is the very reason why only a shorter segment was found), it is difficult to establish whether these clusters can be safely merged. Therefore, the number of found hits does not necessarily fully correspond to the number of unique text reuse.

3.1 BLAST evaluation

As there is not a feasible manner in which to directly estimate the recall of the system on this data, we compare our system to *Passim*, a popular tool for text reuse detection (Smith et al., 2014) used in many similar studies previously, so as to establish a relative comparison to the state-of-the-art. We form a dataset of 2,000 randomly selected documents from the NLF corpus, apply both systems to it, and for every document, we calculate the fraction of its text that was identified as text reuse, with a text length minimum set to 100. The results are summarized in Table 1, and demonstrate a substantial recall gain of the BLAST-based method.

We can see that the BLAST-based method

vastly outperforms Passim in terms of recall. In order to establish that this gain in recall is not at the expense of precision, we sample clusters both randomly and at the very bottom of BLAST similarity scores still acceptable for inclusion in the results and manually verify the proportion of those that are true positives. The proportions are shown in Table 2. The results naturally depend on the length of the texts in the cluster, with shorter texts less likely to be correct hits than the longer ones, given a constant alignment score.

Range	BLAST Precision random	BLAST Precision low	BLAST Coverage	Passim Coverage
300 - 350	1.00	1.00	0.108	0.076
250 - 299	1.00	0.94	0.120	0.078
200 - 249	0.94	0.94	0.133	0.079
150 - 199	0.92	0.86	0.154	0.080
100 - 149	0.86	0.70	0.177	0.080

Table 2: The precision and coverage of the BLAST method on 50 clusters of varying text hit lengths, sampled randomly and at the lowest alignment scores acceptable.

To understand to what extent the hits identified as text re-use are dissimilar, we randomly selected 1000 clusters which contain only two hits of at least 300 characters in length. We then calculate the pairwise character alignment between these two hits and measure the proportion of matching characters, i.e. not gaps nor misalignments. As shown in Figure 2, the alignment values range from around 99% down to as low as 40%, with the bulk of the data in the 70–90% range. For the most part, the repeated texts thus differ in 10–30% of positions, but the difference can be as much as 60%. Partly, these are cases of e.g. advertisements which differ only in numerical values, but partly these are in fact fully identical texts with a massive OCR error rate.

The gain in recall comes at the expense of compute time, with BLAST being about three orders of magnitude slower than Passim. Applying BLAST to the entire NLF dataset required around 150,000 CPU-core hours. This is certainly out of reach for a single computer, but well within modern cluster computing resources, especially since the historical text collection is static and the run only needs to be carried out once.

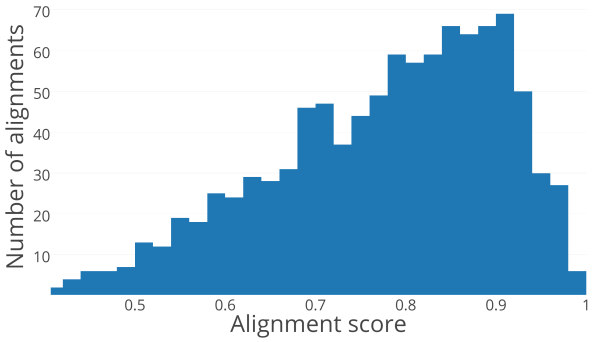

Figure 2: The distribution of alignment scores (horizontal axis) and the number of clusters out of 1000 with the given alignment score (vertical axis). Minimum text reuse length is 300.

4 Text Reuse Clusters – Qualitative Analysis

Copying and reprinting texts from other newspapers took three different forms, which is why we need to differentiate between text reuse, long-term reuse and virality. First, the majority of reprinted clusters consists of advertisements and notices, official announcements and ecclesiastical material. Second, many clusters include old news items, anecdotes, stories and poems that are suddenly reprinted many decades – sometimes even a hundred years – later. This second group is an example of longitudinal text reuse. Finally, the third group is viral news proper. The amount of viral news increases towards the end of the nineteenth century and these texts are often reprinted very rapidly within a short time frame.

4.1 Advertisements and announcements

The first group of clusters, advertisements and announcements, might be interesting sources in their own right for specific research questions. But above all, the changes in their amount tell us a lot about the scope of the public communication network and its historical development year by year even if we completely overlook the content of shared texts. For instance, by importing all text clusters as nodes and edges to a network analysis software, one is able to produce visualizations of the development of relationships between the newspapers and show what were the most dominating nodes in the network.

4.2 Longitudinal text reuse

Because of the wide time frame of the NLF dataset, long-term text reuse opens up an important perspective on historical memory. For instance, the Fennomans were an influential group in the nineteenth-century publicity of the Grand Duchy of Finland. The papers published around the turn of the nineteenth century often reprinted old articles and shorter quotations that supported their cause. To give an example of this, the Table 3 shows the reprints of a patriotic student song "Ännu på tidens mörka vågor" that was probably written by Gustaf Idestam (1802–1851) and first printed in *Åbo Morgonblad* in 3 March 1821 – a newspaper edited by the polemical Romanticist author Adolf Ivar Ardwidsson (1791–1858). The lyrics of the song were then reprinted three times in 1891, crediting the Swedish humorous magazine *Söndags-Nisse* as their source in addition to Arwidsson's paper. The song is also described in *Nya Pressen* as the favourite anthem of the Turku students before the introduction of "Vårt land", the later national anthem. We have found many other similar examples that show the way in which much earlier historical texts were reused for political purposes, opening up an important research question for the strategies of Finnish nationalism.

One example explicitly connected with the state of the Finnish press is the closing down of Arwidsson's newspaper *Åbo Morgonblad* by the officials in October 1821. The reasons for this act were political since Arwidsson had been calling for the wide freedom of the press in his paper. In the last issue of *Åbo Morgonblad* Arwidsson published the document on the official decision for the act. In 1891, 70 years later, this text was reprinted by five different newspapers. The first reprint was in *Åbo Tidningar* (30 September 1891), which used the censorship case of 1821 to discuss the state of the press freedom in 1891 – the censorship law was tightened in Autumn 1891. Other newspapers continued this discussion by reprinting the document from 1821 and also commenting the state of the censorship in 1891. This way the reuse of old news item offered a way to discuss and criticize the situation of the press freedom in 1891.

4.3 Viral news

The third group of text reuse are the actual viral news. According to our preliminary survey of the clustered NLF newspaper corpus, their amount increases rapidly after the The Crimean War (1853–1856). For instance, a bank robbery in Helsinki broke the news in 20 newspapers in 1906. This item was disseminated very rapidly in the Finnish-language press, as is shown in the Table 4. Only in six days it traveled from the urban communication hubs like Helsinki, Turku, Viipuri and Tampere to smaller towns in Ostrobothnia, Savonia, Karelia and Lapland. The viral chain served the need to rapidly tell about a current incident, although this happened without any particular plan, through the existing network of newspapers.

Cluster	Date	Title
639828	1821-03-03	Åbo Morgonblad
639828	1891-02-20	Nya Pressen
639828	1891-02-20	Folkwännen
639828	1891-02-21	Åbo Tidning

Table 3: Reprints of a patriotic song.

The three categories of text reuse could also overlap. Longitudinal chains, for example, might later on transform into viral texts. Old stories or anecdotes could be reactivated after several decades and reused in an infectious manner. BLAST is effective in revealing these different temporal rhythms of text reuse.

Place	Date	Title
Helsinki	1906-11-07	Uusmaalainen
Helsinki	1906-11-07	Helsingin Sanomat
Turku	1906-11-08	Uusi Aura
Helsinki	1906-11-08	Elämä
Tampere	1906-11-08	Tampereen Sanomat
Turku	1906-11-08	Sosialisti
Helsinki	1906-11-08	Uusi Suometar
Jyväskylä	1906-11-09	Suomalainen
Oulu	1906-11-09	Kaleva
Kuopio	1906-11-09	Pohjois-Savo
Tampere	1906-11-09	Kansan Lehti
Viipuri	1906-11-09	Karjala
Sortavala	1906-11-10	Laatokka
Heinola	1906-11-10	Heinolan Sanomat
Savonlinna	1906-11-10	Keski-Savo
Joensuu	1906-11-10	Karjalatar
Lahti	1906-11-11	Lahden Lehti
Kemi	1906-11-12	Pohjois-Suomi
Kristiina	1906-11-12	Etelä-Pohjanmaa
Lahti	1906-11-13	Lahti

Table 4: Reprints of a bank robbery news.

5 Conclusion

We have presented the use of the BLAST method to analyze text reuse in a massive corpus of historical newpapers of poor OCR quality. We have shown that, given sufficient computational power, the method is capable of identifying reprinted text passages that, due to OCR noise, may differ in up to 60% characters when aligned. Analysis of the clusters discovered by the method provides us with new insights into the magnitude and different types of text reuse, and reveals a number of individual examples of historical interest. As a future work, we will strive to develop a text classifier of the different types and topics of text reuse to be able to provide their quantitative analysis. The software developed to carry out the study will be made publicly available as open-source.

Acknowledgments

The work was supported by the research consortium *Computational History and the Transformation of Public Discourse in Finland, 1640-1910*, funded by the Academy of Finland. Computational resources were provided by CSC — IT Centre for Science, Espoo, Finland.

References

Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. 1990. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, Oct.

Ryan Cordell. 2015. Reprinting, Circulation, and the Network Author in Antebellum Newspapers. *American Literary History*, 27(3):417–445.

Kimmo Kettunen. 2016. Keep, change or delete? setting up a low resource ocr post-correction framework for a digitized old finnish newspaper collection. In D. Calvanese, D. De Nart, and C. Tasso, editors, *Digital Libraries on the Move. IRCDL 2015. Communications in Computer and Information Science*, volume 612. Springer, Cham.

Tuula Pääkkönen, Jukka Kervinen, Asko Nivala, Kimmo Kettunen, and Eetu Mäkelä. 2016. Exporting Finnish Digitized Historical Newspaper Contents for Offline Use. *D-Lib Magazine*, 22(7).

David A. Smith, Ryan Cordell, Elizabeth Maddock Dillon, Nick Stramp, and John Wilkerson. 2014. Detecting and modeling local text reuse. In *Proceedings of the 14th ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '14, pages 183–192, Piscataway, NJ, USA. IEEE Press.

David A. Smith, Ryan Cordell, and Abby Mullen. 2015. Computational Methods for Uncovering Reprinted Texts in Antebellum Newspapers. *American Literary History*, 27(3):E1–E15.

Aleksi Vesanto, Asko Nivala, Tapio Salakoski, Hannu Salmi, and Ginter Filip. 2017. A system for identifying and exploring text repetition in large historical document corpora. In *Proceedings of NoDaLiDa 2017*.