Bauhaus-Universität Weimar
Faculty of Media
Degree Programme Computer Science and Media

Applying the Seed-and-Extend Strategy to Text-Alignment

Master's Thesis

Matti Wiegmann

1. Referee: Prof. Dr. Benno Stein
2. Referee: Jun.-Prof. Dr. Florian Echtler

Submission date: June 6, 2018

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Weimar, June 6, 2018

.....
Matti Wiegmann

Abstract

The alignment of reused passages between documents is a central task when handling large collections. Current alignment algorithms for generic text similarity relations are too heterogeneous and thus impossible to compare and improve. With seed-and-extend, these algorithms can be built with a unified strategy. This thesis provides a model for the seed-and-extend strategy for text-alignment and analyses performance evaluation and optimization of its components. The seeding analysis describes a neighborhood relation between different seeders, called relaxation, which can be used to optimize within the configuration space. The extension analysis first adapts DBScan to cluster similar passages of text and then evaluates self-tuning of the clustering parameters for different text similarity relations. Results for seeding show that relaxation can be used to search and optimize in the configuration space of seeding algorithm. Results for extension show that the adapted DBScan can cluster passages of text and hyperparameters can be estimated from the seeds. The conclusion is that, with seed-and-extend, based on relaxation and self-tuning seed clustering, it is possible to do a large-scale automated search for the best text-alignment algorithm.

Contents

Chapter 1

Introduction

A central task when handling large collections of text is the retrieval and extraction of reused text passages between documents, so-called text reuse. The detection of text reuse has applications in plagiarism detection, information flow analyses, and copyright enforcement. Text reuse detection comprises the subproblems of source-retrieval and text-alignment. Retrieval algorithms efficiently find candidate documents which might contain reused text. Text-alignment algorithms detect the specific passages of reused text in a given pair of documents.

The goal of text alignment is to find all pairs of text passages (s, s') that fulfill a target relation $R(s, s')$, where s is a passage in a document d and s' a passage in a document d'. This target relation $R(s, s')$ can range from exact duplication, reordering or paraphrasing to summarization, expansion or plagiarism. For simple relations like exact string or substring matching there are efficient algorithms like Karp-Rabin, Smith-Waterman for local sequence alignment, or Needleman-Wunsch for global alignments. Considering more complicated relations, like summarization, there are no efficient algorithms to find all matching pairs of reused text in a pair of documents.

Seed-and-extend is a sequence-alignment strategy that can be used to detect generic relations. It originated in gene sequence alignment and was used to find functional/ancestral relations or to compensate for sequencing errors. With some modifications, this paradigm can be applied to natural language text (see Figure 1.1).

The seeding step first splits the documents into atomic units of text, like tokens or sentences, and then forms sequences of these units called seed candidates. Then, each sequence from a document d is compared to every sequence from a document d'. If both sequences fulfill a similarity relation, they form a so-called seed. The extension step then merges adjacent seeds to complete alignments, discarding noise and outliers. These alignments reflect the reused

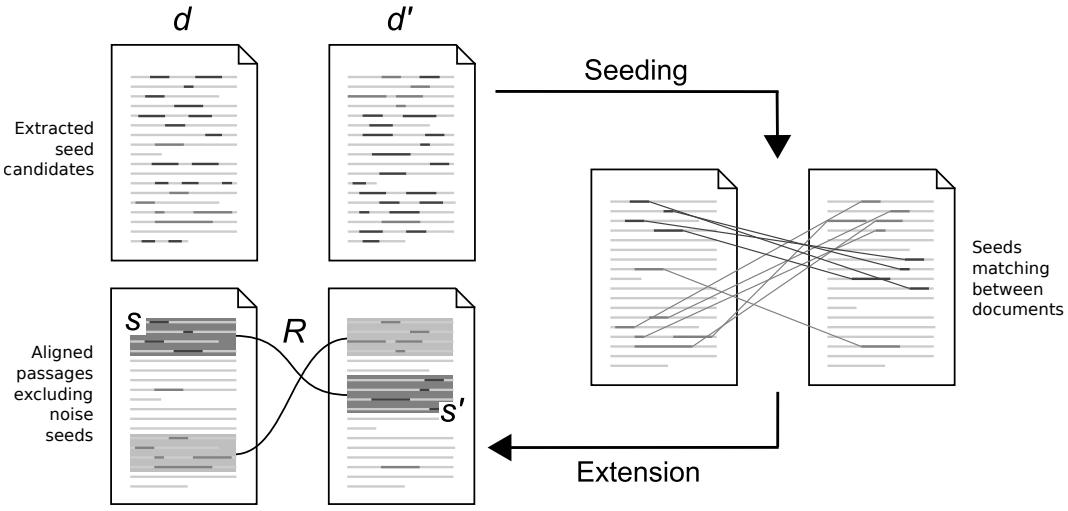

Figure 1.1: Symbolic visualization of the seed-and-extend strategy for extracting aligned pairs of passages (s, s') from documents (d, d')

passages between the two documents. Chapter 2 will review how previous research on text-alignment already employs algorithms that roughly follow the seed-and-extend strategy.

Seeding and extension are usually tightly integrated with those algorithms. Furthermore, the wide range of possibilities to process and compare text allows constructing many different algorithms that employ the seed-and-extend strategy. All of those algorithms will vary substantially in the techniques they use to process text. To gain insights into which parts of the algorithms are responsible for performance, it is not sufficient to only evaluate the alignment results. This is why Chapter 3 introduces a unified model that formulates seeding as a 5-step process of tokenization, filtering, feature extraction, sequencing, and matching. Section 3.1.4 introduces a neighborhood-relation called relaxation based on these components. It will be shown in Section 3.3 that a search in the relaxation-space can be used to optimize the performance of seeders.

In Chapter 4 possible extension strategies will be discussed and it will be explained why density-based clustering is the most promising. A distance function for seeds called boxdistance will be derived. It will be shown that hyperparameters for density-based clustering can be estimated on a per-document basis from the structural properties of the seeds.

Lastly, Chapter 5 serves as an outline for further research based on this thesis. Possibilities to increase alignment performance by combining specialized seeders will be derived in this chapter and an optimization scheme to automatically derive the best possible alignment algorithm will be introduced.

Chapter 2

Background

The seed-and-extend strategy originated, like most sequence-alignment algorithms, in bioinformatics. Seed-and-extend is a general purpose paradigm to find functional/ancestral relations or to compensate for sequencing errors in protein alignment. The goal of this strategy in gene-alignment is to find strings or sequences of acids two proteins share. The seeding step refers to, for example, scanning the acid sequences of two proteins and find matching pairs of sequences. Each matching pair is called a seed. The extension step groups adjacent seeds into similar sequences of maximal length.

This chapter serves as an introduction to the current state of text alignment research. First, the primary literature from the most influential fields, gene alignment and plagiarism detection, will be reviewed. After that, the most cutting edge technologies in the field of text reuse detection will be discussed. Finally, some background on performance analysis for text-alignment will be provided.

2.1 Related Work

The field of string matching and sequence alignment has been extensively researched in both computer science theory as well as bioinformatics. The most comprehensive work on string and sequence matching, indexing, and dynamic programming is *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology* by Gusfield [15]. This book covers most standard algorithms, like Needleman-Wunsch and Smith-Waterman, as well as indexing structures like suffix-trees. The second essential textbook is *Bioinformatics: Sequence and Genome Analysis* by Mount [20]. This book takes a deeper look at specific computational methods, most notably predictions of proteins and protein structure, phylogenetic analysis and fast database search

systems for sequence alignments like FASTA or BLAST. The BLAST algorithm has been adapted for use in information retrieval as eTBLAST [19].

Besides the influences from bioinformatics, text-alignment is a generalization from plagiarism detection, which has been extensively researched in the past. The largest competition on plagiarism and authorship analysis is the PAN competition.[1] PAN originally provided two challenges on plagiarism detection: external and intrinsic, as described by Stein et al. [35]. The external task was about, given a suspicious document, retrieving potential sources from a collection. The intrinsic task focused on analyzing the suspicious document and detecting plagiarism based on style and authorship analysis. In later PAN competitions, the intrinsic detection task has been merged with the authorship tasks and the external task has been generalized to text reuse, with a source retrieval and a text alignment category.

The last PAN including English language text alignment took place in 2014 and provided a set of extensive text reuse corpora and an elaborate evaluation scheme. Both of them will be used in this thesis and are described in detail in Section 2.2 and 2.3. A detailed review of software submissions to PAN and their performance can be found in *Overview of the 6th International Competition on Plagiarism Detection* by Potthast et al. [25]. Another central contribution from this latest instance of the PAN competition is the theoretical derivation of the seed-and-extend strategy for text-alignment. The authors also provide a rough classification of how the submissions fit this paradigm.

From the 11 submissions to the 6th PAN in the text-alignment task, Oberreuter and Eiselt [21], Gillam and Notley [12] and Palkovskii and Belov [22] do not provide sufficient publicly available information to reconstruct the details of their algorithms. Therefore, they will be omitted in the following. Algorithm details like preprocessing, that are common to all approaches, will also be omitted. An overview of the algorithms analyzed for this thesis is given in Table 2.1.

The best performing run was submitted by Sanchez-Perez et al. [28]. Their alignment software uses term frequency * inverse sentence frequency (tf-isf) vectors as atomic units. Tf-isf is similar to tf-idf but regards each sentence as a single document and only the two documents (d, d') as the corpus for calculating the vocabulary and inverse. Two tf-isf vectors are marked as similar if both cosine similarity and dice coefficient are above a threshold of 0.3. Additionally, the algorithm applies a custom divisive clustering approach for seeds. First, the clustering merges all subsequent seeds within a broad gap threshold into what the authors denote as fragments. Each fragment is then repeatedly divided into smaller ones, until every fragment left passes a similarity thresh-

[1] PAN keeps a history of all past tasks and their respective results at http://pan.webis.de.

old. All resulting fragments are then retrieved as aligned pairs of passages if they are longer than 150 characters.

Glinos [13] introduce a modified version of the Smith-Waterman algorithm. Their version is able to detect multiple matches and join adjacent subsequences. The modifications also involve matching two set of stop-words that are approximately equal. After extracting the matched substrings, the algorithm discards every token except for the top 20 most frequent words in the document pair which are longer than 5 characters. After that, the software forms bi-grams as seeds from the remaining tokens. Finally, these seeds are merged if less than 15 characters apart from each other. Clusters of less than 40 characters of length in both documents are discarded.

Shrestha et al. [31] apply an algorithm with two seeders. The first seeder exactly matches token bi-grams, the second one applies the TER-plus [33] metric for machine translation similarity to sentences. Both sets of seeds are combined and seeds are merged when they are closer than a fixed character distance.

Rodríguez Torrejón and Martín Ramos [26] use a version of sorted skip-grams, which they call surrounding context n-grams and odd-even n-grams. These special skip-grams are supposed to be robust against translation and paraphrase obfuscation. The algorithm matches pairs of n-grams if they are exact identical. Afterwards, the software applies a so called granularity filter, which combines matching pairs within a certain distance from each other. Similarly, Gross and Modaresi [14] use skip-grams which match exactly but use single-link clustering to merge adjacent matches below a certain distance threshold. Gross and Modaresi [14] also remove alignments with less than 15 characters in length.

The algorithm submitted by Kong et al. [17] extracts all sentences from the candidate documents and matches them, similarly to Sanchez-Perez et al. [28], based on cosine similarity. Additionally, the algorithm matches two sentences if large parts of the containing words are identical. The authors used this matching system since PAN 2012 and now vary the cosine threshold after detecting the type of reuse in the respective documents.

Abnar et al. [1] used several types of n-grams: traditional n-grams, skip-grams, stop word n-grams and expanded n-grams. Expanded n-grams are synonym-based substitutions of the original n-gram and is meant to capture paraphrase obfuscation. The software matches two n-grams based on transformation distance. This technique views two n-grams as a bipartite graph with edges connecting each one word in the first n-gram with each word of the second one. Each edge has a weight with the probability that one word can be replaced by the other one. The similarity between two n-grams is repre-

Table 2.1: Overview of results and component choices of PAN 2014 submission. Submissions are ranked by the PAN competitions performance metric plagdet (see Section 2.3).

Team	PlagDet	Units	Matching	Extension
Sanchez-Perez [28]	0.878	sentences	vector cosine	divisive
Glinos [13]	0.859	token n-grams	approximate match	agglomerative
Shrestha [31]	0.844	sentences and tokens	TER-p and exact match	agglomerative
Torrejon [26]	0.829	token n-grams	exact match	agglomerative
Gross [14]	0.826	token n-grams	exact match	agglomerative
Kong [17]	0.821	sentences	vector cosine	agglomerative
Abnar [1]	0.672	token n-grams	approximate match	density
Alvi [3]	0.659	character n-grams	exact match	agglomerative

sented by the normalized maximum-weight match for this graph. The resulting matches are then clustered using DBScan with a fixed parameter setting.

Alvi et al. [3] use character 20-grams and the Rabin-Karp algorithm for matching strings between suspicious and source document. The algorithm merges all exact matching passages if they are less than 200 characters apart in the source document and less than 100 in the suspicious document. The software then discards alignments of less than 200 characters in total length.

Several relevant trends can be observed from these state-of-the-art detection algorithms (see Table 2.1). Notable is that all participants roughly follow the seed-and-extend paradigm in that their algorithms first generate matching pairs of text fragments, like token n-grams or sentences, and then merge them afterwards. As unit choices, only token n-grams and sentences are represented, with none being obviously better regarding the final result. Matching on sentences is most commonly done using vector based metrics like cosine similarity, with the exception of Shrestha et al. [31] using a metric for machine translation. Matching on n-grams is more diverse, including exact substring matches, approximate matches of exact subsequences (like in Glinos [13]) or exact matching of approximate subsequences (like in Abnar et al. [1]) or combinations of those. Chapter 3 will provide a more in-depth analysis and comparison of seeding approaches.

Most participants use some form of hierarchical clustering to extend matching passages into alignments. This means, merging the closest neighboring seeds if they exceed a certain character distance threshold. Notable exceptions are Sanchez-Perez et al. [28], who use divisive clustering and Abnar et al. [1], who use density-based clustering. Chapter 4 will provide a more in-depth analysis and comparison of extension approaches.

Considering the seeding step, there are two notable general trends: firstly to include adaptive parameter selection for thresholds and token sequence sizes and secondly to use dynamic programming to adapt seeding procedures to the

Table 2.2: Statistics of the PAN13-test and PAN13-training corpora. This table shows how many labeled pairs of documents are in each category, how long each reused passage is on average and how many reused passages are in each labeled pair of documents on average.

		PAN13-training				
Category	Total	No Plagiarism	No Obfuscation	Random Obfuscation	Translation Obfuscation	Summary Obfuscation
number of pairs	5185	1000	1000	1000	1000	1185
avg. length of passage	1318	0	1090	977	1050	5735
avg. nr. of passages	0.773	0.0	1.252	1.267	1.250	0.201
			PAN13-test			
number of pairs	5185	1000	1000	1000	1000	1185
avg. length of passage	1331	0	1054	987	1095	5944
avg. nr. of passages	0.779	0.0	1.206	1.292	1.308	0.199

respective documents. Shrestha et al. [31], for example, combine a token-based exact match seeder to detect exact relations like non-obfuscated plagiarism and a more relaxed sentence-similarity-based seeder to detect paraphrases or summarizations. Chapter 5.1 will provide a more in-depth analysis of seeder combinations and combinatorial optimization.

All of the software submissions described above were evaluated with the corpora and the evaluation measure provided by the PAN competition. Both of them will be used in this work. Besides the fact that there are no comparably sophisticated resources for scientific analysis of text reuse, using the PAN corpora and evaluation measures allows for comparison with the PAN submissions.

2.2 Evaluation Corpora

In total there are five English language PAN text reuse corpora available [2], two for training and three for evaluation. Since the first test corpus `PAN12-Test` is only a subset of documents of `PAN13-Test` and the third one `PAN14-Supplemental` does not cover all reuse types, only `PAN13-Test` will be used. Similarly, only `PAN13-Training` will be used when necessary. Each corpus features 5 sub-categories for different reuse relations, as described by Potthast et al. [24]:

- `No plagiarism`, where there is no reuse in the document pairs. This category acts as a penalty for false positive detections.

[2] All corpora are publicly available at http://pan.webis.de/data.html

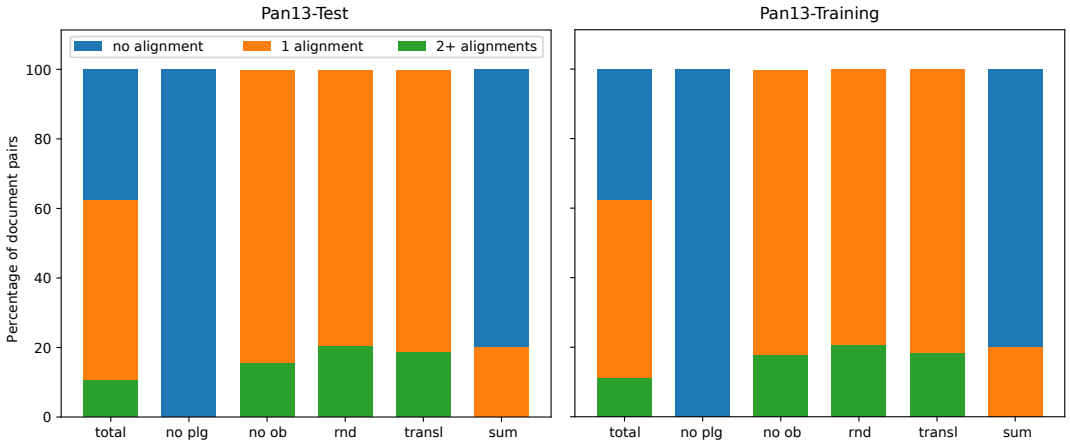

Figure 2.1: Distribution of the number of annotated reused passages within a pair of documents for different categories. The left graph shows the distribution in `PAN13-Test`, the left graph in `PAN13-Training`. Each bar shows the distribution of documents with none, one and more than one reused passage for one category. This graph shows two essential observations: (1) only few document pairs in the corpus have more than one alignment and (2) *summarization obfuscation* category has a very different distribution than the other ones.

- `No obfuscation`, where the annotated passages are identical in both documents.
- `Random obfuscation` is meant to test robustness against machine obfuscation. It is comprised of random text operations like shuffling, addition, deletion, synonym replacement and/or paraphrasing of words or short passages.
- `Cyclic translation obfuscation` is created using a random sequence of machine translations from English into different languages like German, Spanish, Arabic or Japanese and back to English.
- `Summarization obfuscation` is created by injecting a summarization from the Document Understanding Conference (DUC) 2001 corpus [3] into documents from the DUC 2006 corpus.

The corpora contain all raw text source and suspicious documents, a list of candidate pairs of documents and the true alignments. This truth denotes the document identifiers, the category of reuse and for each alignment start character position and length in characters for both documents. Table 2.2 shows the exact numbers for these.

[3]The public DUC corpora can be found at https://duc.nist.gov/data.html.

A notable inconsistency within the dataset is the difference between summary obfuscation and the other categories. Note that the summarization category is created differently compared to the other ones. The documents for summarization are sampled from a news article corpus, while the other categories also use text extracted from web documents and other sources. Another important detail of this discrepancy is the difference in length of the reused passages. There is no difference between the no obfuscation, random obfuscation and translation obfuscation categories on the average length of the reused passages, but summarizations are (naturally) longer. These differences in length and origin have been used, for example, by [3] to distinguish between those categories and adapt their approach. Another important characteristic is the average number of reused passages per document pair (see Figure 2.1). In total, about 40 percent of annotated documents pairs have no reused passages. About half of them in the no plagiarism category and the other half in the summarization category. Additionally, only about 10 percent of document pairs contain more than one annotated reuse and less than one percent 4 or more.

Especially the expected length and number of passages in a document may introduce a considerable bias towards discarding short passages. This can be noticed especially when analyzing the extension modules of the participants' submissions, where the majority discards short passages to improve results. This is in part refined into detecting summarizations and adapting thresholds for merging and discarding accordingly.

2.3 Evaluation Metrics

In addition to the corpora, the PAN competition also provides a comprehensive series of metrics [25] to evaluate the performance of detection algorithms. For the sake of comparability, these metrics will also be used for performance evaluation in this thesis.

A true reused passage in a pair of documents is defined as a reuse case $s = \langle s_{susp}, d_{susp}, s_{src}, d_{src} \rangle, s \in S$. Here, S denotes the set of all reuse cases in the corpus. d_{susp} and d_{src} are references to the documents, where one is labeled as the source and the other as the suspicious document. s_{susp} and s_{src} denote the character positions of the reused passages. Each seed s can be represented by its starting character position and length in both documents as $\mathbf{s} = (s_{src}^{start}, s_{src}^{len}, s_{susp}^{start}, s_{susp}^{len})$ Similarly, a passage that has been found by the algorithm is defined as a detection case $r = \langle r_{susp}, d_{susp}, r_{src}, d_{src} \rangle, r \in R$ and represented by $\mathbf{r}$. A detection of s by r is defined as

$$r \quad detects \quad s \Leftrightarrow \mathbf{s} \cap \mathbf{r} \neq \emptyset,$$

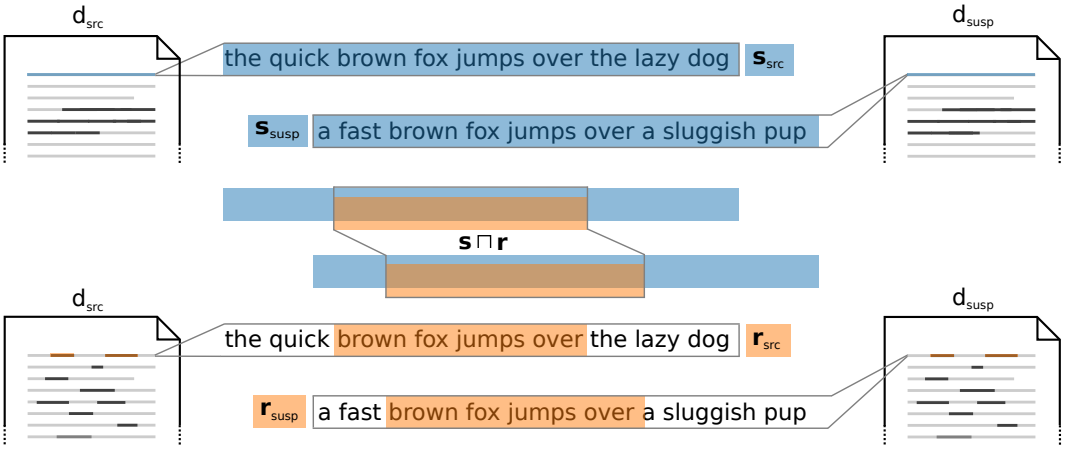

Figure 2.2: Visualization of an example pair of documents d_{susp} and d_{src} and reuse and detection cases.

and the character sequences s_{susp} overlaps with r_{susp} and s_{src} with r_{src} (compare Figure 2.2). The strength of the detection is measured in the length of the set of unique overlapping characters in the relating passages in source and suspicious documents, denoted as $|\mathbf{s} \sqcap \mathbf{r}|$, where

$$\mathbf{s} \sqcap \mathbf{r} = \begin{cases} \mathbf{s} \cap \mathbf{r}, & \text{if r detects s} \\ \emptyset, & \text{otherwise} \end{cases}$$

Precision and recall for a detection algorithm can be calculated over all cases based on the definition of overlap:

$$prec(S, R) = \frac{1}{|R|} \sum_{r \in R} \frac{|\bigcup_{s \in S}(\mathbf{s} \sqcap \mathbf{r})|}{|\mathbf{r}|}$$

$$rec(S, R) = \frac{1}{|S|} \sum_{s \in S} \frac{|\bigcup_{r \in R}(\mathbf{s} \sqcap \mathbf{r})|}{|\mathbf{s}|}$$

The formulas above describe the macro-averaged variations, where recall and precision is first calculated for each reuse and detection case and then averaged over all cases. The alternative is micro-averaging, where all overlapping character positions are unified and recall and precision are calculated once over this union. Macro-averaging is in general more robust, while micro-averaging is more precise. The disadvantage of micro-averaging precision and recall is that longer passages have a much higher impact on the total result. Since the length of the passages in the corpus show a high deviation depending on category (see Figure 2.1), missing a passage in summarization would have a much higher impact on the total.

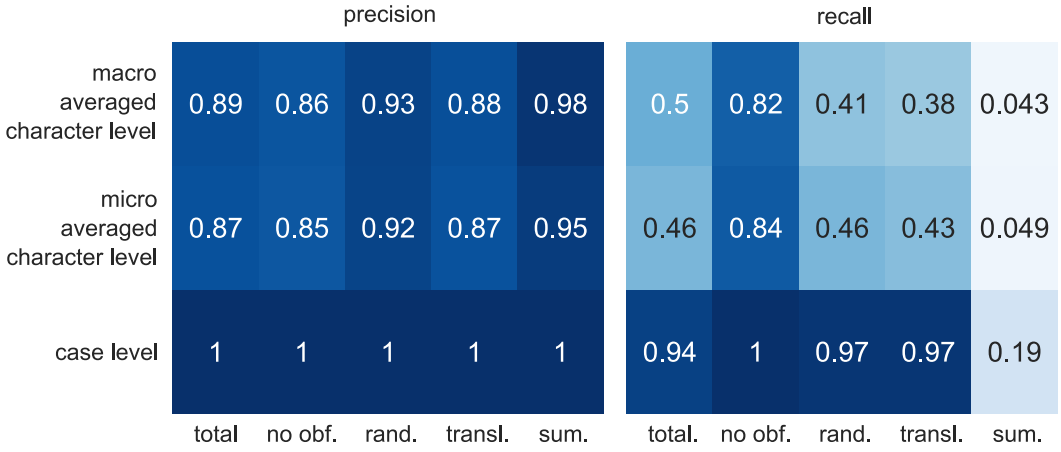

Figure 2.3: Comparison of different possibilities to calculate recall and precision. For this example, the Picapica text alignment algorithm was used. For case level, τ_1 and τ_2 are set to 1.

Recall and precision can also be defined based on case and document-level. A reuse case has a case-level recall of 1 if the ratio of detected to missed characters (the character-level recall) is above a threshold τ_1 and 0 otherwise. A detection case has a case-level precision of 1 if the ratio of missed to total characters of that case (the character-level precision) is above a threshold τ_2 and 0 otherwise. The total case-level recall and precision is the average of these values over all detection and reuse cases in the corpus. Document-level performance is averaged over all documents, where a document is defined as detected if at least one case within the document is detected following the definition of case level performance. See Potthast et al. [25] for more details on alternative performance measures. Figure 2.3 shows a comparison of macro, micro, and case-level performance. For this example, the text-alignment implementation from Picapica [23] was used. Picapica is an online text-reuse analysis tool. Its text-alignment module also follows the seed-and-extend paradigm and uses token 5-grams for seeding and a density-based clustering on 2D representations of seeds.

Case and document-level performance is most useful with a τ of 0, where it can be used to show how many cases overlap somehow with produced seeds. For τ larger than 1, case-level performance is but a more forgiving variation of character-level performance. It is however very likely that most seeders based on smaller units like tokens find at least one seed if there is any noticeable similarity between passages. Thus, case and document-level performance are usually irrelevant (compare Figure 2.3). If not specified differently, recall and precision in this thesis always refer to macro-averaged character-level.

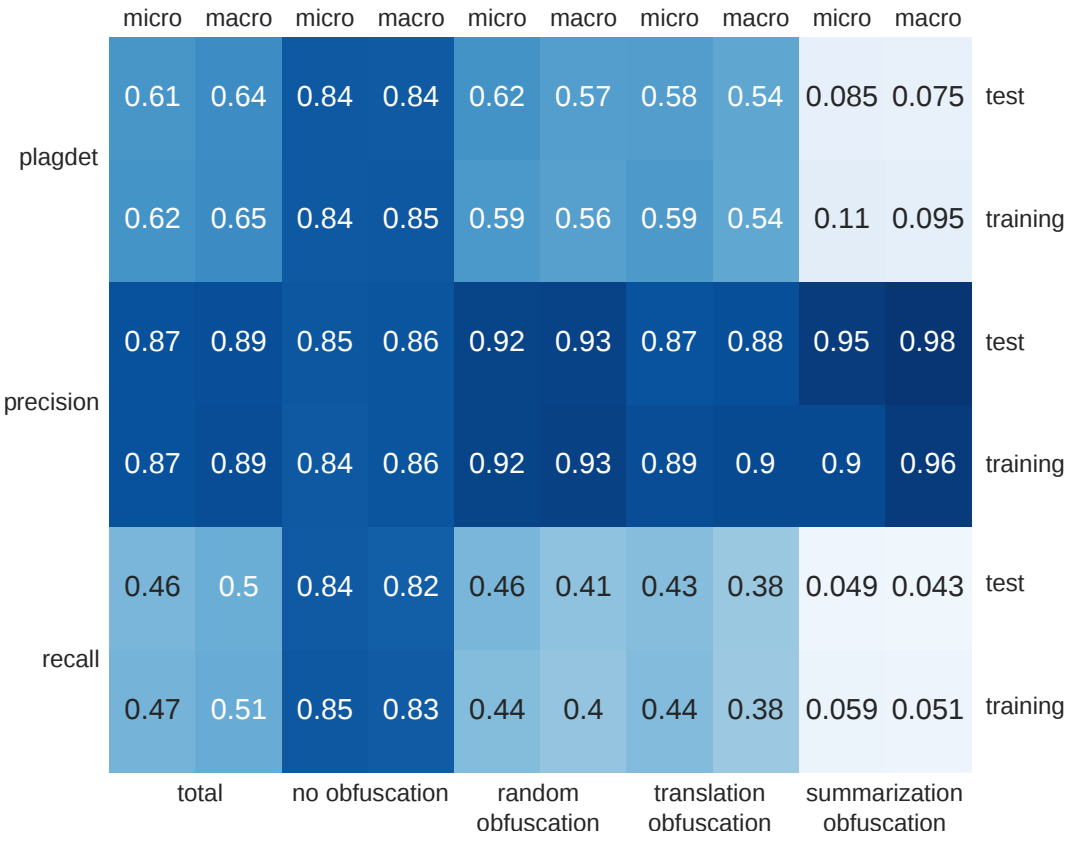

Figure 2.4: Complete extrinsic evaluation of Picapicas text alignment module. The graphic shows micro and macro-averaged plagdet, precision and recall for each category and both corpora.

Besides recall and precision, the PAN evaluation also considers how many detections are needed to cover a reuse case. This term is called granularity and is quantified as:

$$gran(S, R) = \frac{1}{|S_R|} \sum_{s \in S_R} |R_s|$$

$S_R \subseteq S$ is the subset of reuse cases detected by detections in R, and $R_s \subseteq R$ are all detections of a specific reuse case s. The combined evaluation metric for alignment performance is called plagdet[4] and calculated as the harmonic mean of recall and precision (F_1), weighted by granularity:

$$plagdet(S, R) = \frac{F_1}{log_2(1 + gran(S, R))},$$

[4]The original evaluation script is available at http://pan.webis.de/sepln09/pan09-code/pan09-plagiarism-detection-performance-measures.py. This software has been largely reused for plagdet computation, with some modifications for performance.

$$where\ F_1 = 2 * \frac{prec(S,R) * rec(S,R)}{prec(S,R) + rec(S,R)}.$$

In conclusion, the the complete impression of performance of a text reuse detector is given by precision, recall, and plagdet, in the micro and macro-averaged variant, for both corpora and split by categories. Figure 2.4 shows this more complete evaluation on Picapicas alignment algorithm. It can be seen that the difference between both corpora is negligible in terms of insight gained. Also the difference between micro and macro-averaging is rather insignificant. Thus, in addition to the discussion above, only macro-averaging and the `PAN13-test` corpora will be used, unless noted otherwise. This reduces the number of scores to evaluate to a manageable level. The performance of Picapicas alignment module also hints on some of the challenges discussed later. This algorithm matches all identical word-5-grams and is thus rather effective in terms of detecting none-obfuscated reuse. It can also be seen that precision is very high, so this algorithm does not mismatch frequently, following the exact matching done. Compared to the PAN participants this algorithm would score at the lower end, slightly below Alvi et al. [3] (compare Table 2.1).

Chapter 3

Seeder Evaluation

The analysis of the current state of text-alignment in Section 2.1 reveals that token n-grams and sentences are the go-to atomic unit sequences so far. Moreover, matching these sequences is primarily based on the identity of units and approximations like cosine similarity between frequency vectors. However, the performance of the seeders or its components can not be compared based on the performance of the whole text-alignment algorithm. This follows from the fact that, at least in the competition results, there is no clear tendency towards any of these components being notably better in the end.

To compensate this lack of separation, this chapter describes how to uniformly model and evaluate the seeding step. This model of seeding is, in theory, capable of deriving every possible relation between passages from two given documents. In turn, an implementation of this model is capable of analyzing the strength and weaknesses of very different seeders and compare their performance. The primary goal is to determine what the important properties of seeders are and how these can be used to improve detection of reused passages.

3.1 Model of Seeding

As mentioned in Chapter 1, a seed-and-extend algorithm consists of a seeding and an extension step. The seeding-step constructs seeding candidates and matches them to form seeds. The extension step merges adjacent seeds to alignments and removes noise seeds (see Figure 3.1).

To construct seeding candidates, both documents d and d' are first transformed into lists of atomic units $\mathbf{d}$ and $\mathbf{d}$', typically linear in the length of d and d', using a seed heuristic h. These atomic units can be one of several structural units of preprocessed text, from characters, tokens or sentences to encoded, vector or sequenced representations like n-grams. Each position in the lists $\mathbf{d}$ and $\mathbf{d}$' is called a seed key. A matrix M_h is then formed from the

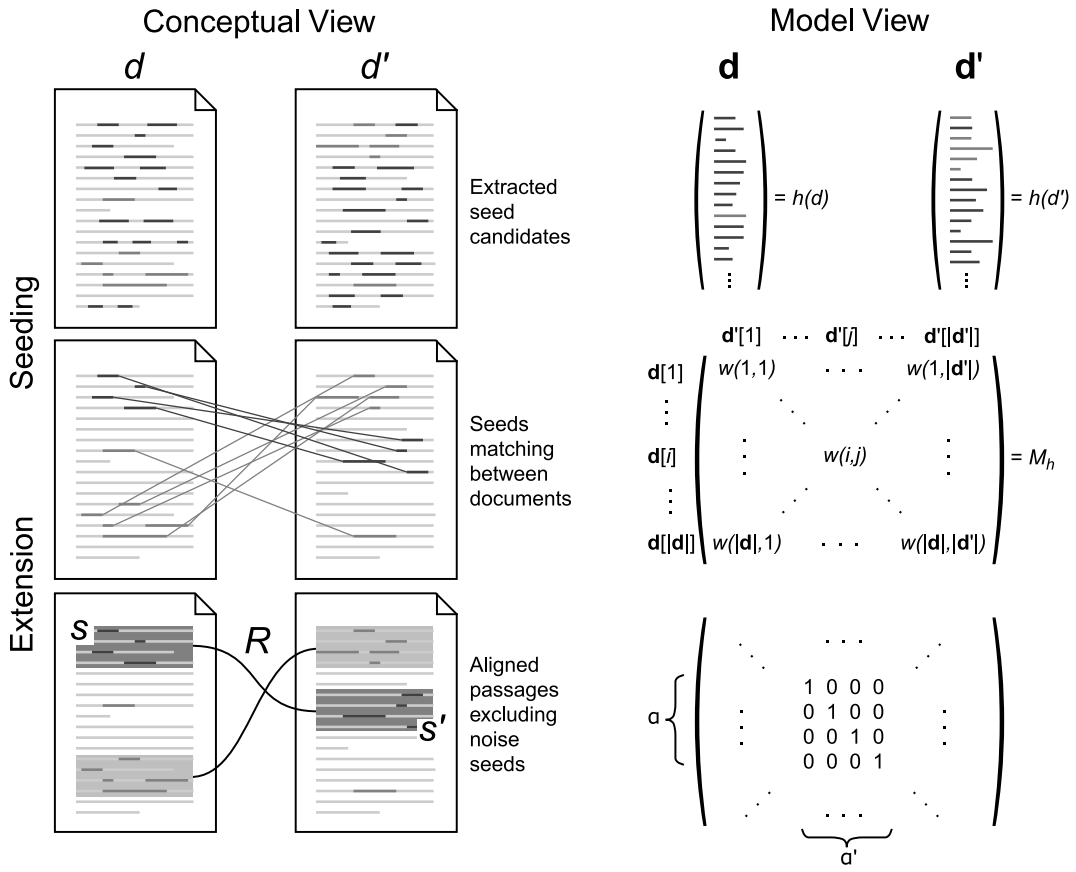

Figure 3.1: Conceptual and model view of a seed-and-extend-based algorithm for detecting aligned pairs of passages (s, s') in documents (d, d')

cross-product of the seed keys of both lists. The i-th row corresponds to the i-th seed key in $\mathbf{d}$, the j-th column to the j-th seed key in $\mathbf{d}$'. Seeds are formed based on this matrix by applying a weight function w to each cell $M_h(i, j)$. The atomic units at the seed keys $\mathbf{d}_i$ and $\mathbf{d}'_j$ are said to match if the weight function $w(i, j)$ exceeds a given threshold. In the most simple case, the weight function is boolean and the units match if they are equal. The weight function accounts for the target relation R. The complete module responsible for extracting candidates and forming seeds is in the following called a seeder and denoted with φ.

The second part of the strategy is the extension step, which takes the weighted matrix M_h as input and outputs a set of pairs of subsequences (s, s') of $\mathbf{d}$ and $\mathbf{d}$' that correspond to the aligned passages of the documents d and d'. Chapter 4 focuses on extension.

In a more abstract view, a seeder consists of a transformation function h, where $h(d) = \mathbf{d}$, such that $|\mathbf{d}| = c*|d|$ and $|\mathbf{d}'| = c*|d'|$ and a matching function

w, where $w(\mathbf{d}_i, \mathbf{d}'_j) = m_{i,j}$, with $m_{i,j} \in M_h$, $\mathbf{d}_i \in \mathbf{d}$, $\mathbf{d}'_j \in \mathbf{d}'$, $i \in \{1, ..., |\mathbf{d}|\}$, and $j \in \{1, ..., |\mathbf{d}'|\}$.

3.1.1 Transformation Function

In layman terms, h combines all operations needed to transform any document d to a vector representation $\mathbf{d}$, such that $\mathbf{d}$ has a constant linear length relationship c to d depending on h. Please note that c is not required be strictly constant. Assume the most primitive instance of h to be a whitespace tokenizer, then the length of $\mathbf{d}$ is the number of all whitespace separated tokens in d, while the length of d is the number of all characters in d, so $c = \frac{|\mathbf{d}|}{|d|}$. In this scenario, a c for d and a c' for d' would only be equal if the distribution of the length of the words in d and d' is equal, which is unlikely. However, assuming that the documents approximately represent the language they are written in, these properties approximately follow this language and c is approximately equal for all documents. This inaccuracy of the definition of c can be ignored since c is only meant to ensure that a transformation function h transforms each document equally.

In more detail, h needs to provide four operations to transform a document into matchable sequences:

- **Tokenization** of the plain text. These resulting units can be whitespace separated tokens, but also characters, sentences, syllables or others. Tokenization here only includes splitting the text by any criterion, so the concatenation of the tokenized sequences should be equal to the original sequence when ignoring splitting characters.

- **Filtering** of the tokens. This includes, for example, only keeping alphanumeric tokens, only nouns, removing stop words or any combination.

- **Extracting features** from the filtered tokens. This includes all transformations that change the tokens itself, like stemming, but also encodings like Soundex codes and vector representations.

- **Sequencing** the extracted features. This includes, for example, constructing n-grams, skip-grams, and sorted n-grams.

The result of these operations, $\mathbf{d}$, has to fulfill two properties: it satisfies the linear relationship c and the text covered by the matchable sequence $\mathbf{d}_i$ follows the ordering of d. More specifically, the smallest character position of the text in d covered by $\mathbf{d}_i$ is larger than the smallest character position in $\mathbf{d}_{i+1}$ and larger than $\mathbf{d}_{i-1}$. Thus, almost every $\mathbf{d}$ can be the result of the

transformation, including every permutation of characters in d. It is possible for $\mathbf{d}$ to be of length 0, only contain constant entries or only have one entry being the text itself. It is however not possible to contain the second half of the text in d before the first half.

3.1.2 Matching Function

The matching function w assigns a weight $\in [0, 1]$ to each pair $(\mathbf{d}_i, \mathbf{d}'_j)$ of matchable sequences. The interval $[0, 1]$ enables granular matching and, in turn, more sophisticated extension. For this thesis, however, only $\{0, 1\}$ will be considered as possible weights, where 1 represents a matching and 0 a non-matching sequence. This means, that M_h in the following can be assumed to be a binary matrix.

This definition of w allows all relations between matchable sequences, with the most general matching function being the trivial matcher, where $\forall \mathbf{d}_i \in \mathbf{d}, \mathbf{d}'_j \in \mathbf{d}' : w(\mathbf{d}_i, \mathbf{d}'_j) = 1$. The most specific matching function is the impossible matcher, where $\forall \mathbf{d}_i \in \mathbf{d}, \mathbf{d}'_j \in \mathbf{d}' : w(\mathbf{d}_i, \mathbf{d}'_j) = 0$. In case of binary weights, a seed $\varsigma = (s^{min}_{\mathbf{d}_i}, s^{max}_{\mathbf{d}_i}, s^{min}_{\mathbf{d}'_j}, s^{max}_{\mathbf{d}'_j})$ is defined as the text between the smallest and the largest character position of the text covered by $\mathbf{d}_i$ and $\mathbf{d}'_j$ if $w(\mathbf{d}_i, \mathbf{d}'_j) = 1$. If the weights are defined on a scale $w(\mathbf{d}_i, \mathbf{d}'_j) \in [1, 0]$, the definition of seed depends on the extension algorithm. This thesis assumes all weight to be binary, so w can output a set of seeds by character positions directly. Otherwise, the w would have to output the matching matrix. This simplification allows the extender to be unaware of the seeder that produced the seeds and how many different seeders were involved. To simplify the notation, a seed in the following is denoted as $\varsigma = (s, s'), \varsigma \in \Sigma$ and the set of all seeds as $\Sigma = \varphi(d, d')$.

Concluding, a seeder φ derives a set of seeds Σ from two documents and consists of five components: a tokenizer t, a filter f, an extractor e, a sequencer z and a matcher m, so $\varphi = (t, f, e, z, m)$

3.1.3 Seeder Performance Evaluation

The performance of a seeder can be evaluated in two ways: extrinsic and intrinsic. Extrinsic evaluation compares seeder performance after a complete seed-and-extend text-alignment pass, where the extension step is constant and the metrics from Section 2.3 are used. Extrinsic evaluation is not particularly useful, because the extension has an inductive bias and performance of a constant extender varies depending on the structure of the seeds, as will be shown in Chapter 4.

Intrinsic evaluation is also based on recall and precision, like extrinsic evaluation, but ignores granularity. Since the model defines the produced seeds identical to aligned passages, precision and recall can be calculated on the seeds directly. The granularity term can be ignored for evaluating seeds since it would directly correlate with the size of the sequences h produces. Therefore, the combined intrinsic measure for the performance of a φ is the F1 score. Note that a φ_1 scoring a higher F1 than a φ_2 does not imply that this is true after extension.

3.1.4 Relaxation

This chapter analyzes which properties and components of a seeder influence its performance, by how much and what can be done to improve it. Formally, the functional differences of seeders can be described by a half-ordered relation called relaxation. Assume the most relaxed seeder $\varphi_g = (t_g, f_g, e_g, z_g, m_g)$ to always produce the all-ones unit matrix and the most specific seeder $\varphi_s = (t_s, f_s, e_s, z_s, m_s)$ the zero matrix. Assume further that each component of the most relaxed/specific seeder is the most relaxed/specific component. A seeder φ_2 is called a relaxation $\varphi_1 \sqsubset \varphi_2$ of a seeder φ_1 if it fulfills the seed-subset property (SSP). The SSP is fulfilled if

$$\forall d, d' \; \forall \varsigma_b \in \varphi_2(d, d') \; \exists \varsigma_a \in \varphi_1(d, d'),$$

such that

$$s^{min}_{a,\mathbf{d}_i} \leq s^{min}_{b,\mathbf{d}_i} \;\wedge\; s^{max}_{a,\mathbf{d}_i} \geq s^{max}_{b,\mathbf{d}_i} \;\wedge\; s^{min}_{a,\mathbf{d}'_j} \leq s^{min}_{b,\mathbf{d}'_j} \;\wedge\; s^{max}_{a,\mathbf{d}'_j} \geq s^{max}_{b,\mathbf{d}'_j}.$$

Assume two seeders φ_1 and φ_2 to be identical except for one variable component ζ_1 in φ_1 and ζ_2 in φ_2. If it can be shown that φ_2 always is a relaxation of φ_1 for any configuration of identical components, then ζ_2 is a relaxation of ζ_1. From this theory follows that all possible seeders can be half-ordered by relaxation between the most general and the most specific seeder. Therefore, the problem of finding the best seeder can be described as a search problem within the space of configurations. While indisputably fascinating, there are some practical limitations in solving text reuse detection by search in the seeder relaxation space. In theory, there is an ideal seeder that can construct a match matrix M_h that perfectly captures the reuse in a document pair. It is not clear, however, if this seeder can be found since it is unclear how the complete component space looks like. Also, it is not clear how to show that a found seeder is ideal for every pair of documents or for every possible relation. Thus, a search in the space of all known seeders is, in the best case, a heuristic to find the best seeder with incomplete knowledge of components, documents, and reuse

relations. Relaxation is still a useful relation to research at this point because it can be used to improve and refine seeders or as neighborhood relation for heuristic search. Relaxation predicts the following changes on seeder performance: The recall of the relaxed seeder is always equal or higher than that of its predecessor. The fact that recall can never fall in case of a relaxation follows directly from the definition of the SSP. If the specialized seeder already detects certain parts of true reuse cases and the SSP holds, then the relaxed seeder also detects those cases but possibly more. This implication does not hold for precision. Precision of seeders considers the ratio of true to false positives, so the new found passages can have a lower or higher ratio, thus the total precision can be lower or higher.

The increase in seeder performance can refer to increasing recall, precision or both, depending on use case. It may also be advantageous to trade precision for recall, since the extension algorithm may be able to handle the additional noise and/or outliers that may cause lower precision. If the goal is to not miss any reuse cases, for example, heavily relaxed seeders may be better, even if precision is low. Please note that finding the ideal seeder to capture all relations in the corpus is vastly out of scope for this thesis, due to the aforementioned implications of the theory. However, it should suffice for now to prove some fundamental predictions of the model for seeding. This means: If a component can be shown to always relax a seeder, the SSP between a seeder and its relaxation is fulfilled and recall does not fall.

3.2 Method

To test the predictions of relaxation, it is necessary to first implement the model of seeding from Section 3.1, including a set of relaxing components. Then, a series of experiments can be derived and executed on the corpora described in Section 2.2.

3.2.1 Model Implementation

This section describes the implementation of the model described in Section 3.1, specifically the necessary components that will be used in the experiments in Section 3.2.2 and the evaluation in Section 3.3. The extension module will be described in Chapter 4. Not all component implementations mentioned in this chapter are used for seeder evaluation in this thesis but are a prerequisite for extension experiments and future work.

Figure 3.2 shows an overview of the processing pipeline for seeding, starting with the two plain text documents as input and computing a list of seeds. Each

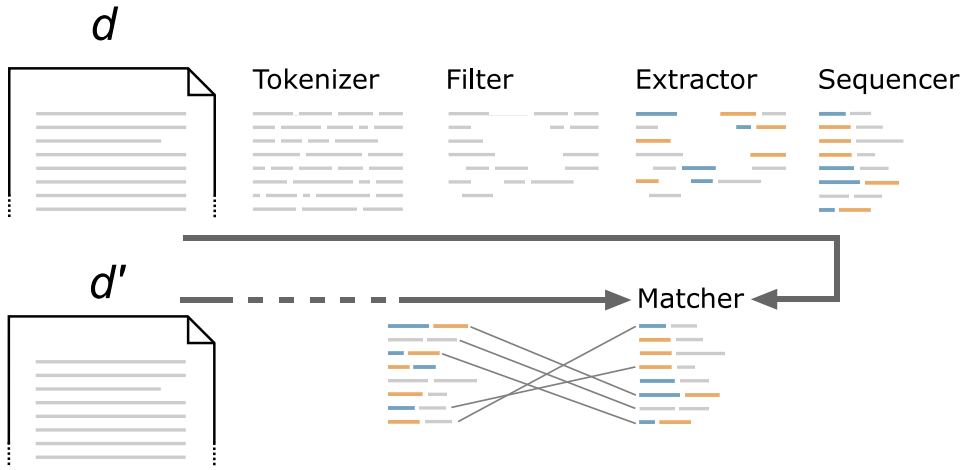

Figure 3.2: Symbolic representation of the text-processing components each seeder is build upon. Both documents d, d' are processed using the same tokenizer, filter, feature extractor and sequencer to compute **d** and **d**'. The matcher then finds seeds from the sequences of both documents.

seed is described as per definition as $\varsigma = (s^{min}_{\mathbf{d}_i}, s^{max}_{\mathbf{d}_i}, s^{min}_{\mathbf{d}'_j}, s^{max}_{\mathbf{d}'_j})$. The computation of Σ is a five-step process through all components of $\varphi = (t, f, e, z, m)$ in that order. Several possible instances of these components have been implemented. For each specific seeder instance, the configuration of these components has to be determined before execution. The following section will describe each component and its parameters for calibration. Please note that the implementation introduces some computational constraints, thus losing some of the generality of the model. This does not impact the experiments and the conclusion.

The first component of each seeder instance is the tokenizer t. The tokenizer takes a plain text document and splits it, depending on configuration, into atomic units of text. These units are then passed to the filter component. Following tokenizer instances are available:

- The `character tokenizer`. Characters are the most basic unit defined, where each character in the document is transformed to a character unit.
- The `whitespace tokenizer` splits text, following the common understanding of tokens, by whitespaces, separates special characters like dots, commas and colons and leaves URLs and abbreviations intact. This component wraps the InfexBA tokenizer from AITools [2].
- The `syllable tokenizer` is based on the word tokens produced by the whitespace tokenizer. The syllable tokenizer uses the AITools wrapper

of David Tolpins TexHyphenator-J[1] [37], which in turn is an implementation of Frank Liang's TexHyphenator [?].

- The `sentence tokenizer` splits the text into sentences. It uses the ArguAna sentence splitter from AITools, which is rule-based, respects abbreviations and targets at well-formatted texts like news articles, which are similar to the documents in the PAN corpora used in this thesis.
- The `paragraph tokenizer` also use an AITools implementation, which splits paragraphs based on double line separators.

The list of tokens output by t is the input for the filter component f. The filter component removes certain units from the list, following its configuration. Following filters have been implemented:

- The `regular expression` filter removes all tokens that do not match a given regular expression.
- The `character-list` filter removes all characters on a list of not allowed characters.
- The `word-token` filter, or just word-filter, removes every token that does not exclusively contain letters.
- The `word-list` filter removes every token that is on a given list of words.
- The `Part-of-Speech` filter removes every token whose POS-Tag is not of the allowed types. Please note that POS-Tags and named entities are determined before tokenization, using the TT4J library[2] [38], which is based on Schmidt [29].
- The `named entity` filter removes every token that is not part of a named entity.
- Boolean operators `AND`, `OR` and `NOT` have been added to combine and negate filters.

The extractor component e takes the filtered list of atomic units and extracts the features which the matching algorithm will be based on. Following feature extractors have been implemented:

[1] TexHyphenator-J was used in version 1.1.
[2] TT4j was used in version 1.2.1.

- The `text` of the given unit is the most basic feature. Text can be extracted as is, as lowercase, truncated (only the first n characters of the unit), stemmed or lemmatized. Stemming is based on the snowball stemmer [36] and lemmatizing is based on TT4J.

- `Part-of-Speech`-Tags and `named entities` as described above. It is also possible to extract the reduced POS-tags as used by Wordnet, namely nouns, verbs, adjectives or adverbs.

- Text with normalized `synonyms` or `hypernyms`. This extractor is based on Wordnet [39] and the JWI library by Finlayson [11]. Wordnet consolidates synonym words into synsets, depending on its part-of-speech. It assigns each word a fixed index within the synset. Thus, synonym normalization means finding the appropriate synset of a word and extracting the word at the first index as a feature. Synsets in Wordnet are connected by hyponym-hypernym relations, which are also indexed. Therefore, hypernym normalization extracts the word token at the first index of the first hypernym-synset. This synset-normalization has some additional configuration options: To use the synset according to a word tokens POS-Tag or the synset of a specific POS-Tag. For example, the word `running` as a noun is in the synset of `track`, as a verb in `operate` and as an adjective in `running`. Additionally, it is possible to normalize hypernyms iteratively. The normalized hypernym of `text` for example is `matter` in the synset `matter, affair, thing`. The normalized hypernym of `matter` is `concern`[3].

- `Term frequency vectors` represent a set of words by their frequency of occurrence. A widely used term frequency vector is the term frequency - inverse document frequency (tf-idf) vector, commonly computed over complete documents. Here, each index represents a word from the collection that can occur in a document, each entry contains the absolute frequency of the word in the document multiplied by the inverse of the frequency of the word in the whole collection. This extractor can either use precomputed inverse term frequencies from an external collection or assume the document pairs d and d' as the complete internal collection. Meaningful vectors can be computed, for example, over paragraphs or sentences.

- `Soundex`[4] is a one-letter and three-number encoding of words or phrases that aims to preserve phonetic similarity. Although it is possible to

[3]Examples are taken from http://wordnetweb.princeton.edu/perl/webwn

[4]Here the implementation of Soundex in the apache.commons.codec package, version 1.8, was used.

encode sentences or longer units, this is not particularly useful since Soundex would be heavily truncated for longer passages of text.

The sequencer component z concatenates the features of the corresponding atomic units to matchable sequences of these features. For this thesis, two sequencers have been implemented: `n-grams` and `skip-grams`. For both n can be freely configured, where n=1 transforms each feature to a sequence of one element. Additionally, the overlap of n-grams and the skip-distance for skip-grams can be configured. The given skip-distance in the implementation includes all lower skip-distances. This means, that each n-gram is contained in every skip-n-gram as the 0-skip.

The matcher component compares each matchable sequence from d with any component from d' and combines them to a seed if they fulfill the matching condition. Implemented matching conditions are:

- `Exact identity`, which is true if each feature at the same position in both sequences is equal.
- `Sorted exact identity`, which is true if two sequences are exact identical after sorting the features alphabetically.
- `Jaccard similarity` is the size of the intersection of the sets of features from both sequences, divided by the size of the union. The matcher based on this is true if the Jaccard similarity is above a threshold.
- `Dice` and `cosine similarity` compare the similarity of vector representations. Two sequences, represented as vectors, match if the dice/cosine coefficient is above a threshold.

3.2.2 Experiments

The following experiments have been constructed to show relaxation of seeders and the predicted effects. Experiments are split into multiple sets, where each set covers multiple relaxations of a seeder by relaxing one variable component multiple times.

The seeders have been implemented using the framework described in Section 3.2.1. One experiment includes running a seeder on every pair of documents in the `PAN13-test` corpus[5]. After computing every set of seeds, the predicted relaxations can be verified. Note that the used corpora contain some inconsistencies, like reuse cases that are not noted in the truth. There might also be some errors in the thirdparty or experiment software. This makes the

[5]This execution has been done in parallel over all document pairs using hadoop 2.7.2 [16].

strict SSP slightly unreliable. To cope with this, the SSP is assumed to be fulfilled if the character-miss-rate (CMR) is below 1%. The CMR is the number of characters covered by ς_1 but not by the relaxed ς_2, divided by the number of characters covered by ς_1. The following five sets of seeders have been derived, each targeting a different type of component relaxation and highlight a different benefit. Table 3.1 shows a comprehensive view of all component settings for the experiments described above and Table 3.2 shows an overview of the important relaxation relationships.

The first set $\varphi_{1,1} \sqsubset \varphi_{1,2}$ is intended to show relaxation through sorting and its effect on performance. The identical components are the whitespace tokenizer, the trivial filter that accepts everything, the lowercase-text extractor, and a 4-overlap-5-gram sequencer. The relaxed component here is the matcher, where $\varphi_{1,1}$ uses the exact identity matcher and $\varphi_{1,2}$ uses the sorted identity matcher. The sorted identity matcher is a relaxation of the exact identity matcher because every sequence pair that matches in its original order still matches after sorting both sequences. The expected result is a slightly lower precision overall and a slightly higher recall on the random obfuscation category. Some of the documents in this category have been obfuscated by shuffling the order of a phrase or passage. From a high-level perspective, relaxing the requirements on the order of a text should lead to better detection of order-based or syntactical obfuscation.

The second set $\varphi_{2,1} \sqsubset \varphi_{2,2} \sqsubset \varphi_{2,3} \sqsubset \varphi_{2,4}$ is intended to show relaxation through approximate matching. The identical components are the whitespace tokenizer, the trivial filter, the lowercase-text extractor, and a 7-overlap-8-gram sequencer. The variable component is the matcher, where $\varphi_{2,1}$ uses the Jaccard matcher with a threshold of 1.0, which is comparable to the sorted identity matcher. $\varphi_{2,2}$ uses the Jaccard matcher with a threshold of 0.75, $\varphi_{2,3}$ with 0.5 and $\varphi_{2,4}$ with 0.25. Jaccard similarity is a relaxation of exact match, so if two sequences match exactly, all features in the sequence are equal. If only 75% of features in both sequences have to be equal and all of them are, the sequences still match. Jaccard similarity has to be more relaxed than sorting from $\varphi_{1,2}$ since Jaccard also ignores the order. The effects on recall and precision should be more significant than for the first set. Precision should decrease the lower the threshold. Recall, on the contrary, should increase in both random and translation obfuscation. On a high-level perspective, relaxing the requirements for identity and order is quite powerful, up to a point. This relaxation captures many types of obfuscation but also can generate many false positives at lower thresholds.

The third set φ_3 is intended to show relaxation through semantics. The identical components are the whitespace tokenizer, the word token filter, a 4-skip-5-gram sequencer and the sorted identity matcher. The variable com-

Table 3.1: Description of the components of the seeders constructed for the experiments, as described in Section 3.2.2.

Name	Tokenizer	Filter	Extractor	Sequencer	Matcher
$\varphi_{1,1}$	whitespace	trivial	lowercase text	4-overlap-5-grams	`exact`
$\varphi_{1,2}$	whitespace	trivial	lowercase text	4-overlap-5-grams	`sorted`
$\varphi_{2,1}$	whitespace	trivial	lowercase text	7-overlap-8-grams	`Jaccard 1.0`
$\varphi_{2,2}$	whitespace	trivial	lowercase text	7-overlap-8-grams	`Jaccard 0.75`
$\varphi_{2,3}$	whitespace	trivial	lowercase text	7-overlap-8-grams	`Jaccard 0.5`
$\varphi_{2,4}$	whitespace	trivial	lowercase text	7-overlap-8-grams	`Jaccard 0.25`
$\varphi_{3,1}$	whitespace	word token	`lowercase text`	4-skip-5-grams	sorted
$\varphi_{3,2}$	whitespace	word token	`strict synonym`	4-skip-5-grams	sorted
$\varphi_{3,3}$	whitespace	word token	`noun-synonym`	4-skip-5-grams	sorted
$\varphi_{3,4}$	whitespace	word token	`strict hypernym`	4-skip-5-grams	sorted
$\varphi_{3,5}$	whitespace	word token	`noun-2-hypernym`	4-skip-5-grams	sorted
$\varphi_{4,1}$	sentence	trivial	tf-isf vector	1-grams	`cosine 1.0`
$\varphi_{4,2}$	sentence	trivial	tf-isf vector	1-grams	`cosine 0.75`
$\varphi_{4,3}$	sentence	trivial	tf-isf vector	1-grams	`cosine 0.5`
$\varphi_{4,4}$	sentence	trivial	tf-isf vector	1-grams	`cosine 0.25`
$\varphi_{5,1}$	whitespace	`trivial`	lowercase text	4-overlap-5-grams	exact
$\varphi_{5,2}$	whitespace	`stopword`	lowercase text	4-overlap-5-grams	exact
$\varphi_{5,3}$	whitespace	`neg.stopword`	lowercase text	4-overlap-5-grams	exact

ponent is the feature extractor, with relaxes synonym relations. The identical components have been chosen to reduce the influence of reordering, insertions, and deletions in the corpus. This should make the word-paraphrasing part more clear. $\varphi_{3,1}$ uses lowercase-text features. $\varphi_{3,2}$ uses the strict synonym-normalized lowercase-text extractor where each token is normalized according to its own POS-tag. $\varphi_{3,3}$ uses the noun-synonym-normalization. This means, this seeder assumes every word to be a noun and uses the first synonym from a words noun-synset for normalization. If there is no noun-synset, because a word has no noun form, then the synset according to the POS-tag is used. $\varphi_{3,4}$ is strict hypernym normalized with one iteration and $\varphi_{3,5}$ is noun-hypernym normalized with two iterations. These synset normalizations have to be relaxations because two identical word tokens have identical normalized synonyms and hypernyms. The effects on recall and precision should be rather minimal. The more relaxed seeders should show slight performance increases on random and translation obfuscation since paraphrasing is part of the construction process of random obfuscation and a translation may also return synonyms. Relaxation effects should be noticeable from $\varphi_{3,1}$ to every other one.

The fourth set $\varphi_{4,1} \sqsubset \varphi_{4,2} \sqsubset \varphi_{4,3} \sqsubset \varphi_{4,4}$ is intended to show relaxation through approximate matching via frequency vectors. The identical components are the sentence tokenizer, the trivial filter, the tf-isf [28] extractor, and the trivial 1-gram sequencer. The relaxed component is the matcher, where $\varphi_{4,1}$ uses the cosine similarity matcher with a threshold of 1, $\varphi_{4,2}$ uses cosine

with a threshold of 0.75, $\varphi_{4,3}$ with 0.5 and $\varphi_{4,4}$ with 0.25. These seeders should behave similarly to set 2, since they match more approximatively with lower thresholds and also ignore the order. However, the reuse cases in the PAN corpora are completely based on complete sentences. This means a sentence based seeder should have notably higher precision than the 8-gram based seeder from set 2. This should be especially notable with more drastically relaxed thresholds.

The fifth set of $\varphi_{5,1}, \varphi_{5,2}$ and $\varphi_{5,3}$ is intended to show the effect filters have on relaxation and performance. Filter are special in that they can fulfill the requirements above and show the effect, but in that this can rarely be shown to be true for all documents and seeder configurations. A seed produced by a seeder with a different filter than the trivial one is defined over the text covered by the matched sequence. If two filtered sequences are matches, all previously removed units between the ones in the sequence are also matched. Also, the filtered-out units between two sequences are never matched, assuming the sequences do not overlap. The identical components in set 5 are the whitespace tokenizer, the lowercase-text extractor, a 4-overlap-5-gram sequencer and the exact identity matcher. The differing component is the filter, where $\varphi_{5,1}$ uses the trivial filter, $\varphi_{5,2}$ uses a stop word filter, that removes all stop words and $\varphi_{4,3}$ is the negated stop word filter, which removes everything but stop-words. The expected effect on performance is that precision is lower for the filtered seeder, notably for the negated stop word filter and recall higher since the sequences overlap. Please note that the trivial filter is more relaxed than the other filters. Also, note that the trivial filter is not the most relaxed filter in theory since the model assumes that the most relaxed filter f_g is always more relaxed than any other. This is not strictly true for the trivial filter, according to the argumentation above.

3.3 Results and Discussion

This section shows the results of the experiments described before. For each set, the predicted relaxation effects should hold. This is indicated by the fulfillment of the SSP or, more precisely, when the CMR is below one percent. Additional observations of the performance of components and seeders will be made when appropriate. Table 3.2 shows the CMR of the predicted seeder relaxations. As predicted, all relaxations except filtering fulfill this criterion. The following section discusses the seeder performance.

The first set of experiments shows the relaxation of sorting the matchable sequences, where seeder $\varphi_{1,2}$ is a relaxation of $\varphi_{1,1}$. The performance can be seen in Figure 3.3. Please note that the *no reuse* category has been omitted in

Table 3.2: Overview of relative CMR of the seeder relaxation experiments. The relative CMR is the average percentage of characters covered by the seeds of the more specific seeder which have not been found by the relaxed seeder.

relaxation	CMR
$\varphi_{1,1} \sqsubset \varphi_{1,2}$	0.0046
$\varphi_{2,1} \sqsubset \varphi_{2,2}$	$6 * 10^{-6}$
$\varphi_{2,2} \sqsubset \varphi_{2,3}$	$1 * 10^{-6}$
$\varphi_{2,3} \sqsubset \varphi_{2,4}$	$5 * 10^{-7}$
$\varphi_{3,1} \sqsubset \varphi_{3,2}$	0.0016

relaxation	CMR
$\varphi_{3,1} \sqsubset \varphi_{3,3}$	0.0030
$\varphi_{3,1} \sqsubset \varphi_{3,4}$	0.0074
$\varphi_{3,1} \sqsubset \varphi_{3,5}$	0.0026
$\varphi_{4,1} \sqsubset \varphi_{4,2}$	0.0173

relaxation	CMR
$\varphi_{4,2} \sqsubset \varphi_{4,3}$	0.0020
$\varphi_{4,3} \sqsubset \varphi_{4,4}$	0.0076
$\varphi_{5,2} \sqsubset \varphi_{5,1}$	0.1208
$\varphi_{5,3} \sqsubset \varphi_{5,1}$	0.3572

the figures but is still used to calculate the total. In a nutshell, precision and recall are about as predicted. Sorting the sequences slightly increases recall by up to two percentage points per category and slightly reduces precision by up to one percentage point. The effect of this relaxation is negligible for this seeder configuration. Also, simple reordering without additional obfuscation is a rare border case in the `PAN13-test` corpus. The performance differences are still not trivial, considering there are about 5.24 million characters in `PAN13-test` part of a true reuse case. So a one percentage point increase in total recall means about 52.400 additionally detected characters, which is roughly up to 52 additionally detected passages.

Another notable observation is the precision of 0.51 in the *no obfuscation* category. This category should only contain identical copies of passages, which φ_1 should be able to almost completely detect. The same effect can be observed in the case of larger n-gram sizes, for example the sorted 5-gram seeder $\varphi_{1,2}$ and the 1.0 threshold Jaccard 8-gram seeder $\varphi_{2,1}$. The sorted skip-4-5-gram seeder $\varphi_{3,1}$ with word-filter mitigates this, however. $\varphi_{6,1}$ (see Figure 3.8) is identical to $\varphi_{1,2}$ but uses the word-filter. This change drastically increases precision by up to 35 percentage points on *no obfuscation* but with a slightly lower recall. $\varphi_{6,3}$ is also identical to $\varphi_{1,2}$, but replaces the 4-overlap-5-grams with 4-skip-5-grams. The sorted skip-grams also increase precision by up to 15 percentage points on *no obfuscation* with increased recall. If precision increases significantly when only consider all-letter 5-grams, then there are many false positives involving 5-grams with numbers, punctuations, diacritics and so on. This phenomenon frequently occurs with unclean web-content extraction. If a web-content extractor removes all HTML-tags, for example, this may leave sequences of special characters like commas or colons. The PAN corpora have partially been created from web collections, so this is a likely explanation.

The hypothesis for the second set of experiments is that a lower threshold for the Jaccard similarity implies a relaxation of the seeder. For 8-grams based seeders, a threshold decrease of 0.25 means two fewer units need to

Table 3.3: Description of the components of φ_1. The variable component is the matcher.

Name	Tokenizer	Filter	Extractor	Sequencer	Matcher
$\varphi_{1,1}$	whitespace	trivial	lowercase text	4-overlap-5-grams	`exact`
$\varphi_{1,2}$	whitespace	trivial	lowercase text	4-overlap-5-grams	`sorted`

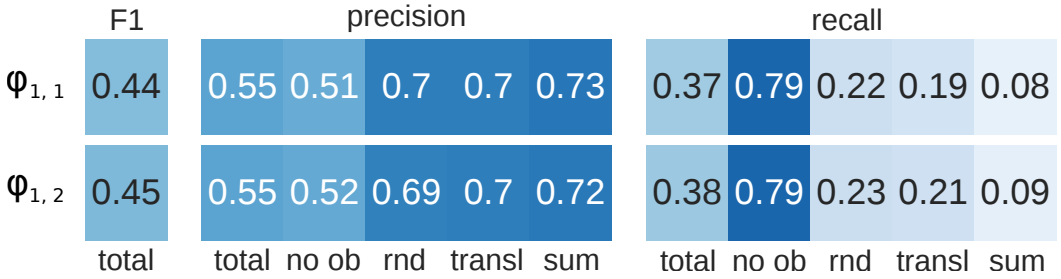

Figure 3.3: Results of the first set of experiments with a focus on relaxing the order requirements for matching. $\varphi_{1,1}$ is the 5-gram exact match seeder, $\varphi_{1,2}$ relaxes the matcher to sorted identity.

match exactly for the sequence to match. The performance changes displayed in Figure 3.4 show the effects. The predicted increase in recall is, as expected, quite notable. Approximate matching that also ignores order is one of the most powerful relaxations. It equals the most general matcher at the lowest threshold and therefore relaxes all relations. The results also show that there is an optimal threshold for φ_2, with regards to F1, somewhere between 0.5 and 0.25. Notable is the drastic drop in precision from four of eight required token matches in $\varphi_{2,3}$ to two in $\varphi_{2,4}$. Also, the most notable increase in recall happens at the *random* and *translation obfuscation* categories, but not so much for *summarization*. These two observations mean that the random and translation obfuscation techniques leave, on average, a little less than half of the tokens in an eight token radius intact. For *summarizations*, however, the F1 decreases about ten percentage points when decreasing the Jaccard threshold from 0.75 to 0.5. Considering that the extension may be robust to some of these false positives, the increase in recall may still be worth it. It should be clear that just relaxing the requirements for matching will not solve summarization detection. The last notable observation in this set of experiments is that reducing the requirements for matching from 1.0 notably increases precision, which seems unintuitive. The most likely cause for this is again the text extraction done while creating the corpora, as described in the discussion of set 1 above.

The third set of experiments shows the effects of relaxing the semantics of single words by normalizing synonym words to a uniform one or to a common

Table 3.4: Description of the components of φ_2. The variable component is the matcher.

Name	Tokenizer	Filter	Extractor	Sequencer	Matcher
$\varphi_{2,1}$	whitespace	trivial	lowercase text	7-overlap-8-grams	`Jaccard 1.0`
$\varphi_{2,2}$	whitespace	trivial	lowercase text	7-overlap-8-grams	`Jaccard 0.75`
$\varphi_{2,3}$	whitespace	trivial	lowercase text	7-overlap-8-grams	`Jaccard 0.5`
$\varphi_{2,4}$	whitespace	trivial	lowercase text	7-overlap-8-grams	`Jaccard 0.25`

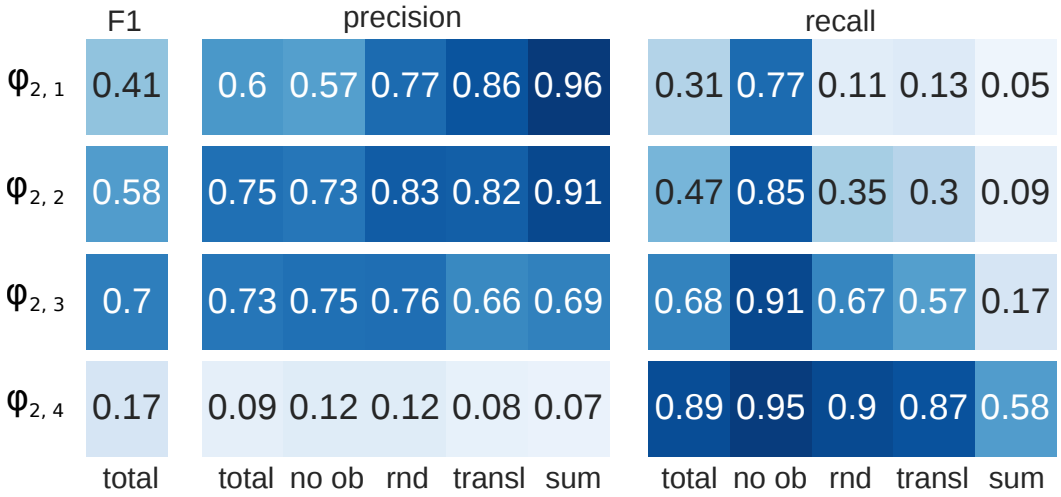

Figure 3.4: Results of the second set of experiments, showing relaxation of order and identity with Jaccard similarity. The seeders use token 8-grams and relax the threshold of the Jaccard similarity matching in steps of 0.25.

hypernym. The evaluation results can be seen in Figure 3.5. Because the PAN corpora mix different obfuscation strategies to generate reuse cases, the identical components have been chosen to mitigate ordering and word-insertion and deletion effects. Also, all non-word tokens have been filtered out to avoid negative side effects of the corpus creation. A notable observation here is that the up to four percentage points gain in recall is lower than expected but about double the gain of simple sorting as done in set 1. Also, this increase is not completely compensated by a drop in precision, especially on the most relaxed two iteration noun-hypernym normalized $\varphi_{3,5}$. The low increase in recall overall is likely based on a lack of purely word-level based paraphrasing in the corpus.

The fourth set of experiments show the relaxation of the cosine similarity threshold on tf-isf vectors. The tested thresholds are 1.0, 0.75, 0.5 and 0.25. Note that frequency vectors are always in the positive space, so the cosine is in [0, 1]. These seeders are based on Sanchez-Perez et al. [28] seeding strategy

Table 3.5: Description of the components of φ_3. The variable component is the feature extractor.

Name	Tokenizer	Filter	Extractor	Sequencer	Matcher
$\varphi_{3,1}$	whitespace	word token	`lowercase text`	4-skip-5-grams	sorted
$\varphi_{3,2}$	whitespace	word token	`strict synonym`	4-skip-5-grams	sorted
$\varphi_{3,3}$	whitespace	word token	`noun-synonym`	4-skip-5-grams	sorted
$\varphi_{3,4}$	whitespace	word token	`strict 1-hypernym`	4-skip-5-grams	sorted
$\varphi_{3,5}$	whitespace	word token	`noun-2-hypernym`	4-skip-5-grams	sorted

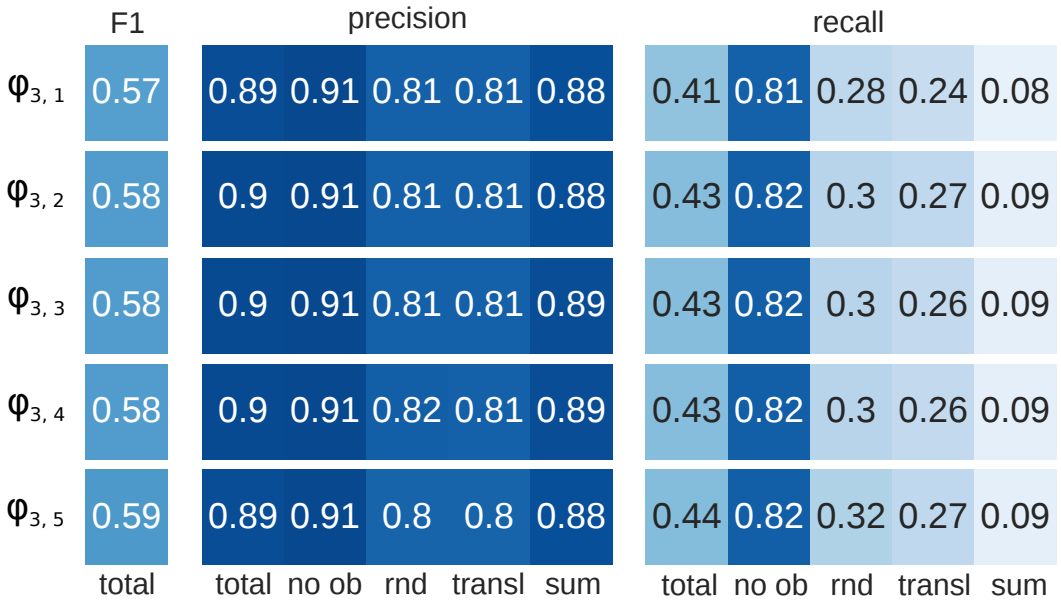

Figure 3.5: Results of the third set of experiments. All seeders use 4-skip-5-grams and sorted identity matching to eliminate ordering, insertion and deletion effects of the obfuscation on the comparison. The variable component is synonym and hypernym normalization.

and are quite similar in effect to the approximate matching seeders from set 2. The cosine based seeder φ_4, as seen in Figure 3.6, has lower recall but higher precision with comparable thresholds than seeder φ_2. The likely reason is that φ_4 compares complete sentences and the reuse cases have been constructed based on sentences. Also notable is that cosine similarity falls less steep with lower matching requirements compared to Jaccard similarity.

Observe that recall for $\varphi_{4,1}$, which only matches identical sentences, is about 28 percentage points lower on *no obfuscation* cases than that of $\varphi_{2,1}$. This is the reverse effect to the low precision in this category shown by other seeders without word-filter. If the non-obfuscated reuse cases are not completely equal, the token-based $\varphi_{2,1}$ can still detect the exact parts of these sentences while

Table 3.6: Description of the components of φ_4. The variable component is the matcher.

Name	Tokenizer	Filter	Extractor	Sequencer	Matcher
$\varphi_{4,1}$	sentence	trivial	tf-isf vector	1-grams	`cosine 1.0`
$\varphi_{4,2}$	sentence	trivial	tf-isf vector	1-grams	`cosine 0.75`
$\varphi_{4,3}$	sentence	trivial	tf-isf vector	1-grams	`cosine 0.5`
$\varphi_{4,4}$	sentence	trivial	tf-isf vector	1-grams	`cosine 0.25`

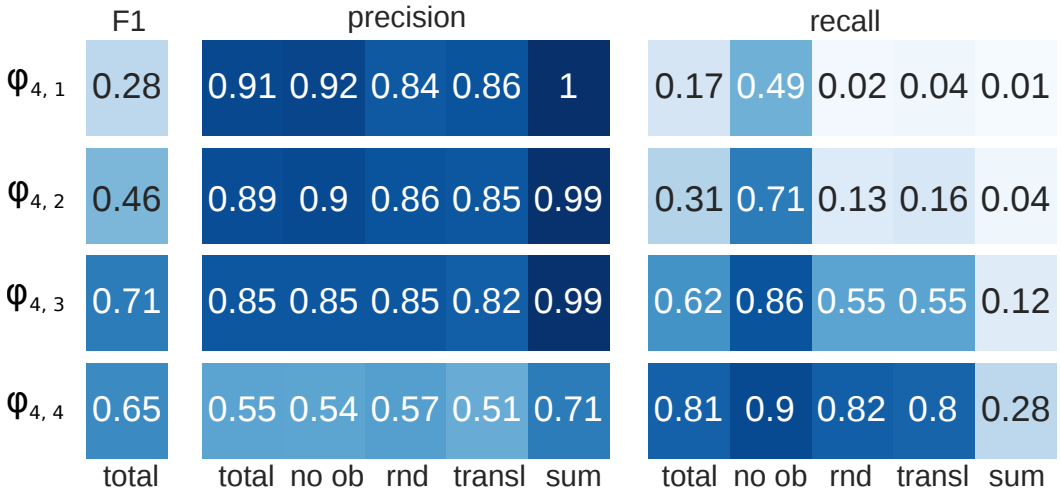

Figure 3.6: Results of set 4 of experiments, showing relaxation of order and identity over sentences with cosine similarity of frequency vectors. The seeders are based on cosine similarity on tf-isf vectors and relaxation of the cosine threshold needed for matching.

$\varphi_{4,1}$ can not. There is also the possibility that some documents in the corpus are not completely well-formed, so the sentence splitter does not work precisely.

The premise of the fifth set of experiments is that the neighborhood in the relaxation space of all seeders can be predicted by component relaxation. This can be used to guide a heuristic for optimization of seeders but does not necessarily predict the whole relaxation space. In layman terms, a component replacement can relax one specific seeder, but not necessarily all. Replacement of components is not the defining factor of the relaxation space. This effect is more difficult to utilize for a search. It is possible that a relaxing replacement can only be shown empirically by applying and evaluating the results.

In the designed experiments, replacing the stop-word filter with the trivial filter is not a relaxation, considering the relative CMR of 12 percent (see Table 3.2), even though φ_5 uses overlapping n-grams. The results of the fifth set of experiments are shown in Figure 3.7. Also remember the effects of the

Table 3.7: Description of the components of φ_5. The variable component is the filter.

Name	Tokenizer	Filter	Extractor	Sequencer	Matcher
$\varphi_{5,1}$	whitespace	`trivial`	lowercase text	4-overlap-5-grams	exact
$\varphi_{5,2}$	whitespace	`stop word`	lowercase text	4-overlap-5-grams	exact
$\varphi_{5,3}$	whitespace	`neg.stop word`	lowercase text	4-overlap-5-grams	exact

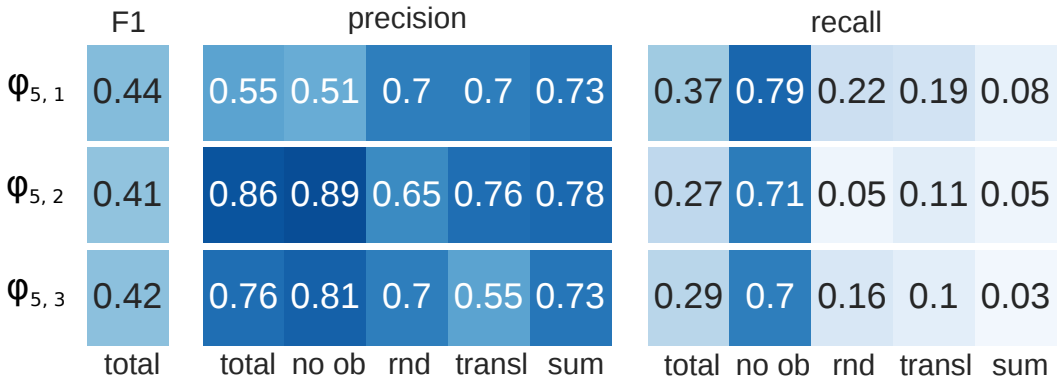

Figure 3.7: Results of set 5 of experiments about filtering, showing that seeders can be relaxations, even if the components do not strictly require them to. Here, the seeder $\varphi_{5,1}$ without filtering is a relaxation of the other two, where $\varphi_{5,2}$ remover stop words and $\varphi_{5,3}$ removes all non-stop words.

word-token filter and how it applying it reduces recall since it specializes the seeder.

The last notable observation over all experiments is that none of the designed seeders is particularly effective on the summarization obfuscation category. The highest recall here is 0.58 by $\varphi_{2,4}$, which in turn has a precision of 0.07. In other words, this seeder likely matched a lot of passages quite liberally and managed to include some passages in this category. While it can be assumed that the extender can compensate for false positives, as will be discussed in Chapter 4, it is unlikely to work on this scale with only this one seeder. Summarization is difficult because of the difference in length of the passages. This means, for example, that one sentence in the source document has to match multiple sentences in the suspicious document. In theory, there exists a seeder capable of doing this. Still, a different strategy seems more promising for now: combining seeders. Since the output of a seeder in the model is defined over the character positions in the document, it is possible to combine the seeds output by different seeders into one set as input for the extension. This means, multiple seeders that detect different relations can be constructed and their seeds combined. These combined set of seeds may detect

Table 3.8: Description of the components of φ_6

Name	Tokenizer	Filter	Extractor	Sequencer	Matcher
$\varphi_{6,1}$	whitespace	word token	lowercase text	4-overlap-5-grams	sorted
$\varphi_{6,2}$	whitespace	trivial	lowercase text	4-skip-5-grams	exact
$\varphi_{6,3}$	whitespace	trivial	lowercase text	4-skip-5-grams	sorted
$\varphi_{6,4}$	whitespace	word token	lowercase text	4-skip-5-grams	exact

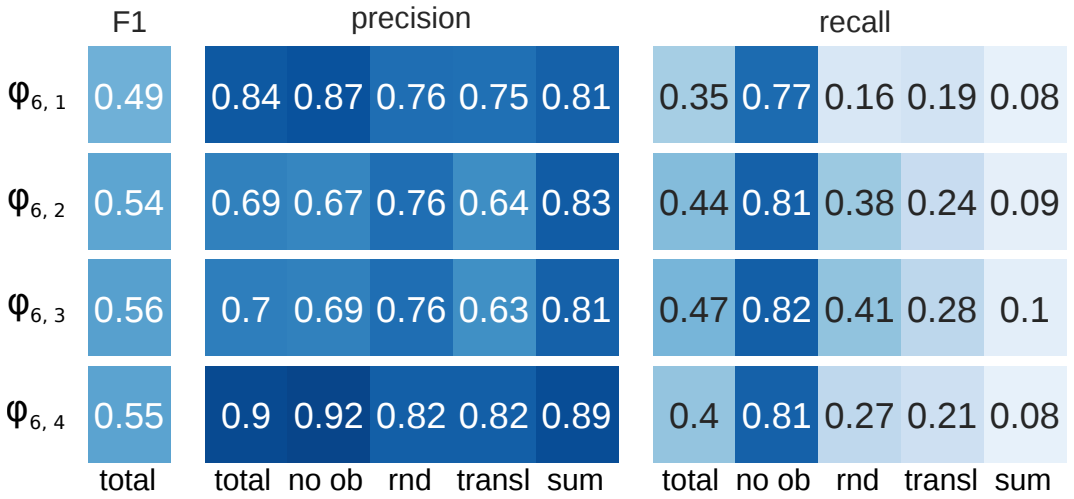

Figure 3.8: Evaluation of the seeders φ_6, as described in Table 3.8. This graphic illustrates the effects of skip-grams and the word-token filter on seeder performance.

difficult relations like summarization more easily. The advantages and pitfalls of seeder combination will be discussed in Section 5.1.

Concluding, this chapter provided detailed insights about seeding in theory and on how to construct and evaluate seeders. This has been done by defining the components of seeders and the relations between them. It should be clear that seeders can be optimized based on relaxation to better capture relations of similarity between passages of text. The following chapter will discuss how to extend the seeds into aligning passages, including their capabilities to close gaps between seeds and filter false positives to find the best matching reuse detection indicated by the seeds.

Chapter 4

Seed Extension

Extension is the second part of seed-and-extend text-alignment. In the general model (see Section 3.1) the extender is defined as a transformation of a weighted matrix M_h to a set of a set of pairs of subsequences (s, s') of **d** and **d'** that correspond to the aligned passages of the documents d and d'.

Section 3.1.2 describes a simplification for binary matrices M_h, where the extender takes a set of seeds Σ as input instead of a weighted matrix. This simplification allows a single extender to accept seeds generated by multiple seeders without any additional knowledge about these seeders. The assumption for extension is, that aligning passages in the documents are structures formed by the detected seeds. This means that, in the model of seeding with binary matching functions, seed extension can be solved using adaptations of common clustering strategies. These adaptations need to handle outliers, noise, artifacts and distance functions for seeds.

This chapter should first introduce different approaches to seed clustering, discuss the problems and limitations specific to the seeds properties. Afterward, the DBScan [10] algorithm will be adapted for seed extension. Finally, the problem of selecting good hyperparameters will be discussed. Then, possible strategies to determine good parameters will be derived: estimating good, fixed parameters for each seeder, optimizing parameters from cluster evaluation and learning parameters from the structure of the seeds.

4.1 Overview of Existing Extension Strategies

As described in Section 2.1, all participants in the latest PAN competition on English language text-alignment use alignment clustering at some point in their algorithms, although not always explicitly. The most common algorithm used is a version of hierarchical, agglomerative single-link clustering, although participants do refer to it usually as distance-based merging. The idea is that

two seeds can be merged if the distance (or gap) between the passages in the source document is below a threshold θ_1 and in the suspicious document below θ_2. The distance between two sequences in the same document is usually the single-link distance between the two closest character position. If the passages overlap, the distance is usually zero. A few participants used more advanced extension strategies. The algorithm by Sanchez-Perez et al. [28] used a variation of hierarchical, divisive clustering. Here, seeds are first merged into one cluster and then divided into smaller ones. This division is repeated until the passages of maximum length are found. The maximum length here means, that there is no longer alignment that still fulfills the cosine-similarity requirement of its frequency vectors. Abnar et al. [1] employed a density-based clustering with the DBScan algorithm.

4.1.1 Discussion of Alternatives

Single-link is the most popular clustering strategy among PAN participants, likely due to its simplicity and effectiveness on the PAN corpora. The problem with the applied versions of single-link is that they need three hyperparameters: the thresholds for source and suspicious document and the threshold for filtering outliers and artifacts. It seems likely that single link only works well in the PAN competitions because the number and the average length of the reused passages follow a rather strict pattern. This can be seen in the analysis of the PAN corpora in Section 2.2. This suspicion is supported by another observation: There is a notable structural difference between the summarization category and the other ones and there is a strong tendency within the PAN submissions to classify the detections into summarization and others and adapt the parameters accordingly. Similarly, the divisive algorithm depends

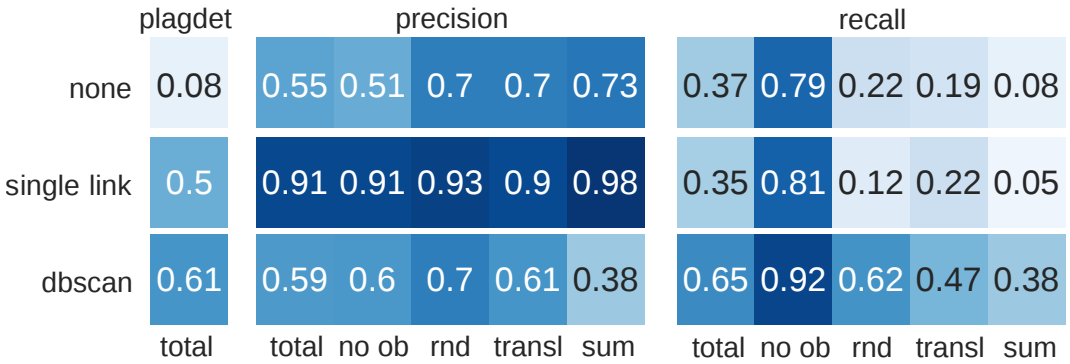

Figure 4.1: Examples of precision, recall and plagdet of different clustering strategies with identical seeders. Granularity of *no extension* (none) is 28 and close to 1 for the other two.

on a fixed similarity measure, which works well on the PAN corpora but may not generalize well. Density-based clustering is the most promising here since it only needs two hyperparameters and can detect outliers and noise without additional thresholds or post-extension filtering. It is also not necessary to determine different parameters for different reuse types. Thus, it should generalize well if the hyperparameters and the distance function are chosen well. Figure 4.1 shows the difference in precision, recall and plagdet with different extension modules. The identical seeder used for these examples is a basic 4-overlap-5-gram seeder with whitespace tokenizer, the trivial filter, and exact matching. The single link variant uses the hyperparameters used by Alvi et al. [3], which is a 200/200 character merge-threshold in source and suspicious document and 200/100 character threshold for post-extension filtering. The DBScan variant uses an epsilon of 1600 and a minPts of 3. The example without extension is the baseline for comparison with low precision and low recall. The plagdet without extension is naturally really low because plagdet is weighted by granularity, as described in Section 2.3. The single-link variant using Alvis parameters has high precision since outliers and noise are filtered strictly by post-processing. Single-link does, however, underperform at closing gaps, which is especially important for exact matching sequences in the more difficult categories. The DBScan, which will be explained in more detail in the next section, performs worse in precision, since outliers are not filtered that strictly, but achieves higher recall. This means DBScan is better at detecting the underlying structures of the alignments.

4.1.2 DBScan for Seed Extension

The original DBScan algorithm [10] clusters points based on density. It uses two hyperparameters, epsilon, and minPts. Each point is a core point if it has at least minPts other points within an epsilon radius. A point that is not a core point but is within epsilon of one is called a border point. A point that is neither core nor border point is called a noise point. Two points are density-reachable if they are within an epsilon distance and one of them is a core point or if there is a path of density-reachable points between them. Two points are part of a cluster if they are density reachable and if both are density-reachable from every other point in this cluster.

A seed is defined as $\varsigma = (s^{min}_{\mathbf{d}_i}, s^{max}_{\mathbf{d}_i}, s^{min}_{\mathbf{d}'_j}, s^{max}_{\mathbf{d}'_j})$, where $s_{\mathbf{d}_i}$ and $s_{\mathbf{d}_j}$ are the complete text in the documents d and d' covered by the two matching sequences. The first and last character index of the sequence in the original document are denoted by s^{min} and s^{max}. In the 2-dimensional space spanned by the character indices of d and d', each seed is defined by the two points (x_{min}, y_{min}) and (x_{max}, y_{max}), where $x_{min} = s^{min}_{\mathbf{d}_i}$, $y_{min} = s^{min}_{\mathbf{d}'_j}$, $x_{max} = s^{max}_{\mathbf{d}_i}$

and $y_{max} = s^{max}_{\mathbf{d}'_j}$. These two points in the 2-D space can be interpreted as lines or boxes (see Figure 4.2)), while the DBScan algorithm is defined using n-dimensional points. DBScan can be adapted in two ways to work with seeds. When interpreting seeds as a line, the Euclidean distance between the two closest points of each seed can be used as the distance function. This strategy works under ideal conditions where seeds do not overlap and are arranged on a line. This is usually not the case, especially not with short or overlapping seeds (see Figure 4.2). Also, DBScan would be required to always compare both points of each seed to find the closest distance, which is impractical. The second possibility is to interpret seeds as boxes that span a space. If two seeds cover the same text in both documents the boxes will overlap. In this case, the distance between two seeds should be 0. If a position in one document matches multiple positions in the other document, there will be multiple seeds overlapping on one axis. Here, the distance should be the difference on the non-overlapping axis. If the boxes do not overlap at all, the Euclidean distance between the two closest points can be used as distance-metric between the seeds. All those cases can be uniformly represented using the box spanned

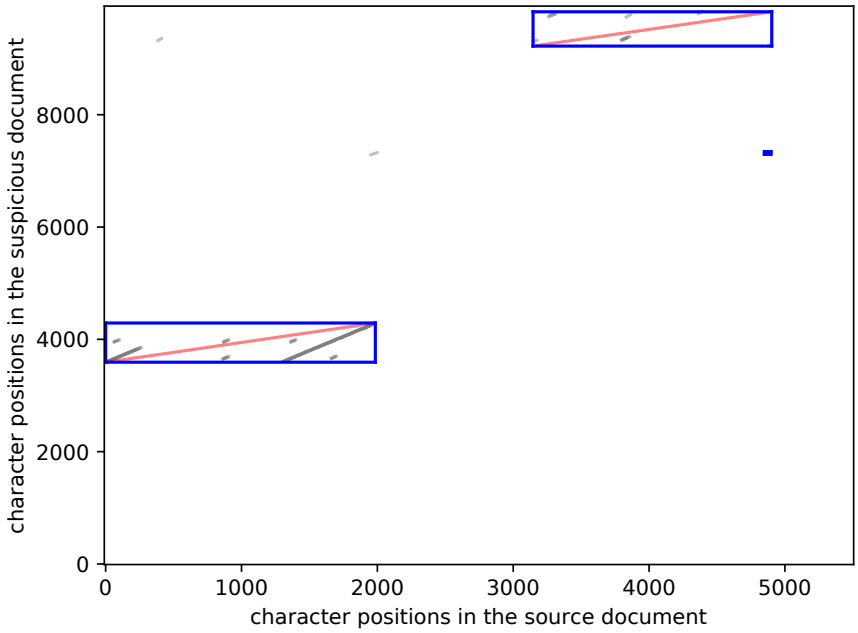

Figure 4.2: Plot of the seed clustering of one example pair of documents. The grey lines are the matching sequences of atomic units (seeds). The darker the grey areas, the more seeds overlap at this position. The line-representation of the resulting alignment is colored red, the box-representation is blue. This example shows the DBscan extender with an epsilon of 1600 and a minPts of 3.

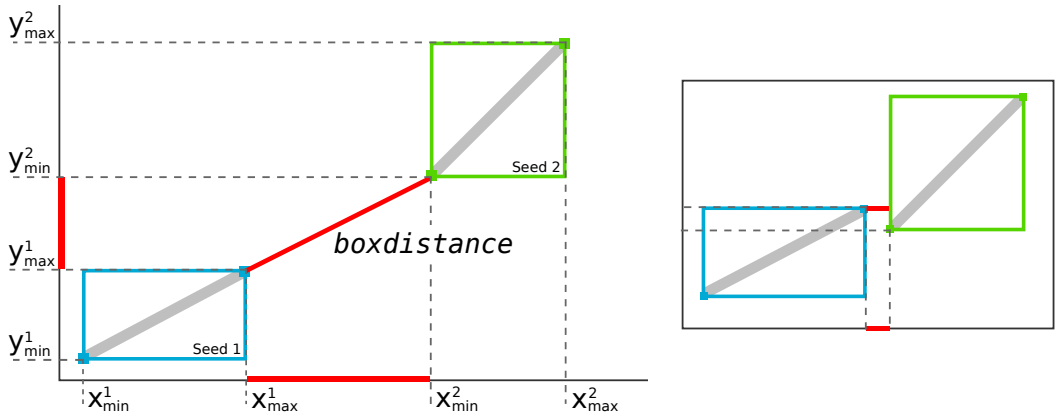

Figure 4.3: Annotated example of the distance function *boxdistance* between two seeds. The *boxdistance* is the length of the diagonal of the box between the closest two non-overlapping points of each seed. The left example shows that *boxdistance* for non-overlapping boxes is just the Euclidean distance between the closest two points. In the right example the boxes overlap at y, so *boxdistance* here is only the x-distance.

between the closest start point of one seed and end point of the other seed (see Figure 4.3). More precisely

$$boxdistance(\varsigma_1, \varsigma_2) = \sqrt[2]{\max_{x \in X} x^2 + \max_{y \in Y} y^2},$$

where

$$X = \{0, x^{\varsigma_2}_{min} - x^{\varsigma_1}_{max}, x^{\varsigma_1}_{min} - x^{\varsigma_2}_{max}\}$$

and

$$Y = \{0, y^{\varsigma_2}_{min} - y^{\varsigma_1}_{max}, y^{\varsigma_1}_{min} - y^{\varsigma_2}_{max}\}.$$

If the boxes only overlap on the x-axis, the box separating the seeds will have no width and the distance will be the difference on the y-axis and vice versa. If the seeds overlap completely, the distance will be 0. If the seeds do not overlap at all, the distance is the Euclidean distance between the closest points of each seed.

The extension module of the text-alignment framework built for this thesis implements DBScan for seeds by wrapping the ELKI [30] implementation of the generalized DBScan. *Boxdistance* has been passed to the ELKI implementation as a custom distance function. In the algorithm, one seed is represented as a 4-dimensional point.

4.2 Hyperparameter Estimation

As mentioned before, DBScan uses two hyperparameters: epsilon and minPts. Increasing epsilon generally closes gaps because it increases the distance for points to become density-reachable so farther away points count towards the cluster. Increasing minPts increases the number of density-reachable points required to form a cluster. This has the effect that higher minPts discards clusters in low-density areas. Both parameters are in a way dependent on each other, so increasing or decreasing both might have, to a certain degree, no effect. For seed extension, increasing only epsilon merges adjacent seed clusters. Increasing minPts discards small clusters which are farther away from other clusters. If epsilon becomes too large compared to minPts, seed clusters that are far away will be merged. This means, all the text between the merged clusters will be part of the alignment. This may be beneficial but will likely introduce many false positives. If minPts becomes too large compared to epsilon the algorithm will discard many small clusters, even if they are close together. These effects can be seen in Figure 4.4.

The selection of parameters depends on the seeds itself. If those seeds are large and overlap is low, like sentence seeds, for example, a low epsilon and low minPts might be best. If epsilon is too high in this example, a sentence in between the seeds which is not covered by a seed itself might be included in the alignment. Non-overlapping sentences also likely require a lower minPts because non-adjacent sentence seeds are much farther away. Overlapping token n-gram seeds require completely different parameters because there are many more seeds in a much closer area. It is therefore essential to find good parameters and there is no sweet spot that works reasonably well for all types of seeds. Additionally, through filters and the combination of seeds from different seeders increases the range of seed structures immensely. Following this, it is necessary to determine the hyperparameters for every extension run, depending on the set of seeds passed to the extender.

4.2.1 Parameter Evaluation

There are two possibilities to evaluate the performance of an extender, given the seeds are static. The first one is internally by comparing the increase in precision, recall and/or F1 through extension between different extenders. The second one is extrinsically by plagdet (see Section 2.3). Plagdet is the better choice because it is the same as F1 but considers granularity. The increase compared to seeder performance is not relevant since the seeders used for evaluation are static.

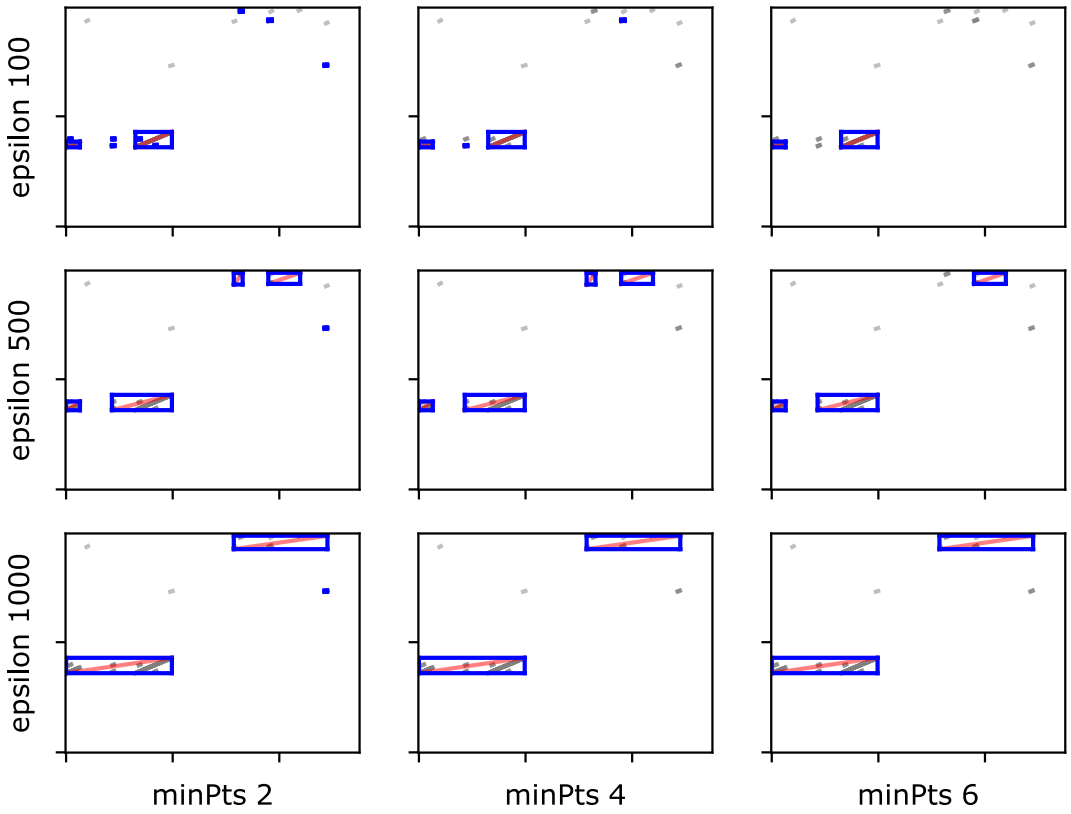

Figure 4.4: This graph shows the different outcomes of seed-clustering with different parameter settings. It can be seen that a larger epsilon increases the are in which seeds are clustered, while a higher minPts discards more sparse and distant areas as noise.

The optimal parameter selection can be found using a grid search. Since initial experiments show the plagdet function to be approximately convex, the grid can be constrained to determine an optimum. Figure 4.5 shows the result of a grid search of the DBscan extender for a simple word-token 4-overlap-5-gram seeder with exact matching. The optimum for this seeder is around a minPts of 3 and an epsilon of 1600. As can be seen in the figure, these parameters do not need to be overly precise. At least for this seeder, any combination of parameters between a minPts of 3-4 and an epsilon between 500 and 2000 will be near-optimal. A grid search is however quite time-consuming. If the goal is to find the best seeder, seeder combination or just determining the selection for many documents, a grid search is too time-consuming. This is why the following sections will discuss how to find these near-optimal parameters without searching the whole grid.

Considering the analysis of the effects of epsilon and minPts in Section 4.2, the plagdet function over the parameter space has to be approximately convex.

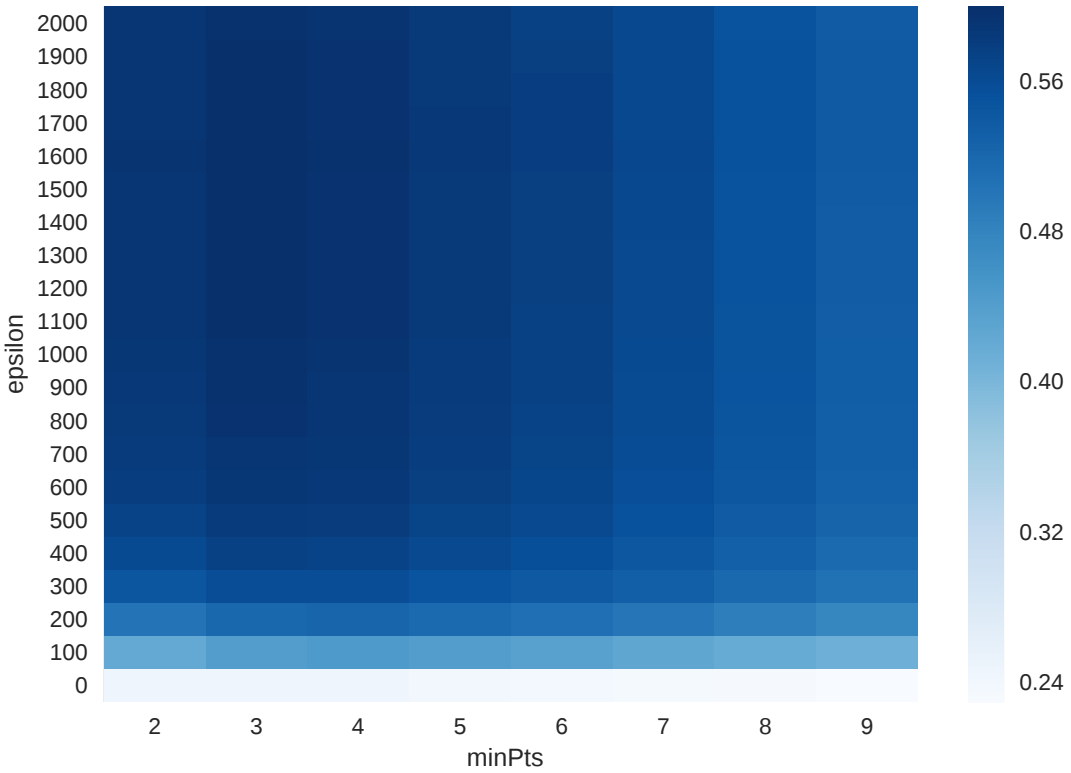

Figure 4.5: Plagdet of DBScan extension over a grid of parameter choices. One seeder was used to generate a static input. This seeder used 4-overlap-5-grams of word tokens and exact matching.

The assumption is that there is a "sweet spot" of both parameters where the plagdet score will be maximal. If any or both parameters become too low or high, the clustering will deteriorate and the plagdet will fall. If the truth is known, the optimum for a static seeder can be found easily using local optimization.

To use the truth is, however, impractical for several reasons. First, a truth may not always be available when comparing algorithms for specific purposes. Secondly, the corpora only cover a selection of possible reuse categories and test and training corpora are very similar in structure. Therefore it is likely that selecting parameters based on the truth will overfit, which should be avoided. The third and probably most important argument is about document versus collection-level parameter selection. Document-level selection refers to determining the best hyperparameters for each document individually while collection-level refer to determining one set of parameters for the whole collection. If parameters are determined via optimization against the truth, they will have to be used for all future documents equally. Consider the results of the grid search from Figure 4.5. Here, the parameters at each

grid point are the same for each document pair in the collection. The best collection-wide plagdet score is 0.601, archived with an epsilon of 1600 and a minPts of 3. Now, if the best performing parameters for each document pair is selected from the grid individually, the collection-wide plagdet is 0.716. It can be expected that, especially when combining seeders with different specializations, document-level parameter selection outperforms collection-level. Note that this optimal document-level plagdet score is an upper bound and determined with a known truth. It likely can not be reached otherwise.

The goal for the following sections is to find out if parameters can be estimated on document-level without consulting the truth and if performance is equal or even better than predetermined parameters.

4.2.2 Local Optimization

One possibility to select parameters on a per-document basis without consulting the truth is to use a convex optimization strategy on an internal cluster validation metric. These validation metrics are commonly used for unsupervised comparison of alternative clusterings and are useful for estimating, for example, the ideal number of clusters. These metrics usually compute a score that either represents favorable structural properties or how good a point fits its cluster. Examples for the first metric are the Dunn or Davies-Bouldin index, an example of the second one is the silhouette index. If one of these metrics becomes optimal in the near-optimal parameter space, it could be used for optimization instead of the truth.

The Dunn [9] index of a clustering C is defined as

$$\text{Dunn}(C) = \frac{\min\limits_{i \neq j}\{d_C(C_i, Cj)\}}{\max\limits_{1 \leq l \leq k}\{\Delta(C_l)\}},$$

where $d_C(C_i, Cj)$ is the distance between the two clusters C_i and C_j and $\Delta(C_l)$ is the length/diameter of a cluster. For seeds, the distance between two clusters is the *boxdistance* and the diameter of a cluster is the length of the diagonal of the box that spans all seeds within. The Dunn index grows larger if the largest cluster shrinks and if the closest clusters move farther away from each other. Optimizing against the Dunn index can produce smaller and more distant clusterings. The problem for seed extension is that Dunn is only defined if there is more than one cluster. In the used corpora, as well as in most real applications, having more than one reused passage is a rather rare case (see Section 2.2). When defining the distance as a very large or very small value, in case there is only one cluster left, the index would become very small/large in

return, so the last remaining clusters would either never or always be merged. Both of those adaptations defy the purpose of the metric (see Figure A.1).

The Davies-Bouldin index [8] is defined as

$$\text{Davies-Bouldin}(C) = \frac{1}{N}\sum_{i=1}^{N}\max_{j}\frac{s(C_i)+s(C_j)}{d_C(C_i,C_j)}.$$

d_C is again the *boxdistance*. $s(C_i)$ is a measure of the tightness or scatter of the cluster i, defined as the average distance of each point within to the centroid of that cluster. Davies-Bouldin is quite similar to the Dunn index but is based on scatter and is not strictly biased towards only the worst sub-structure. It still requires multiple clusters to be defined and is thus similarly unusable for the scenario in this thesis (see Figure A.2).

The Silhouette [27] is originally meant as a tool for visual inspection of how well a point k fits the cluster it is in. The silhouette is calculated as

$$\text{Silhouette}(k) = \frac{\text{md}(k,C_k) - \min\limits_{i}\text{md}(k,C_i)}{\max\left\{\text{md}(k,C_k), \min\limits_{i}\text{md}(k,C_i)\right\}},$$

where $\text{md}(k,C_i)$ is the mean distance of the point k to every other point in the cluster C_i, C_k is the cluster k belongs to and $\min_i \text{md}(k,C_i)\}$ is the mean distance to every point in the neighboring cluster of C_k. The silhouette of a point k is in $[-1,1]$, where 1 means the point fits perfectly in its cluster and -1 it fits perfectly in the neighboring cluster. Averaging the silhouette provides an impression of the tightness of each cluster, which can interpret how well each point fits its cluster and thus how good the clustering is. Amorim and Hennig [4] used this in combination with feature rescaling to better predict the correct number of spherical clusters with noise features. The average silhouette measure is still ill-defined when the most common number of clusters is one or none (see Figure A.3). When using average silhouette as fitness-function for local optimization, this leads to the same conclusion for the one-cluster-case as for Dunn or Davies-Bouldin: either the last two clusters are never merged or the heuristic tries to merge all points, including noise, into one cluster.

In conclusion, internal validity measures work for seed extension as long as it expects more than one cluster. A measure that does not expect more than one cluster is however not useful to compare different clusterings because there will be only one anyways. If there is only one cluster left for a parameter setting, all that can be analyzed is the diameter and scatter of this one cluster and the noise points. It is though pointless to derive a fitness function using only scatter and diameter. Clusters of seeds are very elliptical in nature, so merging adjacent clusters or including noise-points would increase the length

Table 4.1: List of the seeders and components used for learning good DBScan hyperparameters.

Name	Tokenizer	Filter	Extractor	Sequencer	Matcher
$\varphi_{7,1}$	whitespace	word	lowercase text	4-overlap-5-grams	exact
$\varphi_{7,2}$	sentence	trivial	tf-isf vectors	uni-grams	0.3 cosine
$\varphi_{7,3}$	whitespace	only stopwords	lowercase text	2-overlap-3-grams	exact
$\varphi_{7,4}$	whitespace	word	lowercase text	0-overlap-8-grams	0.5 Jaccard

and scatter. Any metric using a linear relationship of length and scatter would either never or always merge clusters and noise-points (compare Figure A.4).

4.2.3 Estimation from Seeds

However, internal validity might not be the only way to suggest a parameter setting for a good clustering. It has been noted before that the parameters depend on the seeds. Sentence-based seeds, for example, work well with low minPts (see Figure A.5). Token n-grams and very relaxed seeders, especially with overlapping seeds, work well with high minPts and large epsilon (see Figure 4.5 and Figure A.6). Those settings avoid artifacts and fragmented clusters more effectively.

Therefore, it should hypothetically be possible to estimate good parameters from the length and number of seeds, following above observations. The idea is if there are few, large seeds, then minPts and epsilon should be rather low and if there are many short seeds, minPts and epsilon should be high. More precisely, the hypothesis is that the hyperparameters have some form of a functional relationship to the distribution of the length of the seeds. A good approximation of this function could be used to calculate the parameters for any seeder or combination of seeders used.

However, the length of the seeds from different seeders do not follow the same distribution (see Figure 4.6), especially not if produced from a seeder combination. This means, the parameter-function cannot be simply inferred from the distribution, although it is possible to learn this function from example seeders.

The experiments in the following chapters should show that this functional relationship exists and that it can be learned from the examples. More precisely, it should be possible to learn from one seeder and predict a different one reasonably well, as long as the length distribution is similar. This should also hold for seeder combinations, in that the function can be learned from the seeders and generalized to the combinations and vice versa. Also, these experiments should show how well generalization to unseen seeders and seeder combinations works.

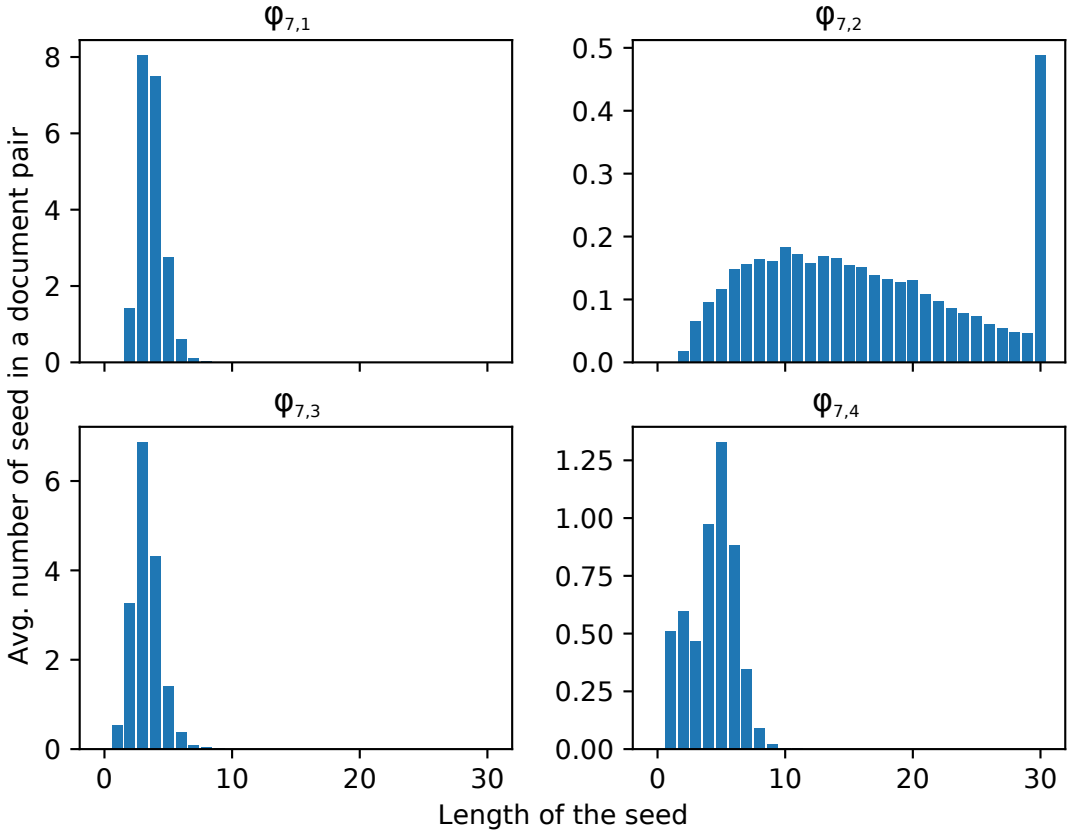

Figure 4.6: Histogram of the seeds length for different seeders (see Table 4.1).

4.3 Parameter Learning

This section describes the algorithms, encodings, and components needed to learn the parameters from the seeds. Section 4.3.1 a set of different seeders to generate examples, a good parameter setting for these examples, features representing the seeds and an appropriate machine learning model. The evaluation scheme is described in Section 4.3.2.

4.3.1 Machine Learning Model

The seeders used to generate the examples determine the quality of the generalization of the model to a degree. Table 4.1 lists the seeders used for parameter learning. It was mentioned before that the learned function over the length distribution should generalize well no matter what seeders are used to learn it. However, a machine learning model can only approximate what can be found in the examples. If the seeders chosen to learn from are biased towards a certain distribution, the model will not generalize well. Compare for example the histogram differences in Figure 4.6 between $\varphi_{7,1}$ and $\varphi_{7,2}$. These seeders have

been chosen to show that a model trained on one seeder generalizes well to a different seeder with a similar distribution of seed length. Observe how $\varphi_{7,1}$ and $\varphi_{7,3}$ have similar histograms. Additionally, if multiple seeders with different histograms are used to train the model then it should generalize better to unseen seeders with a still different histogram.

Each of these seeders has been run on the whole `PAN13-training` corpus to generate the seeds for each document pair. For extension, a grid search has been done as shown in Figure 4.5. This way, the plagdet over the whole corpus could be computed for each combination of epsilon and minPts. The parameters ranged from epsilon from 0 to 2000 in steps of 100 and for minPts from 0 to 9 in steps of 1. This way, there are 168 different clusterings for each pair of documents for each seeder. For each pair of documents, the parameter set with the highest plagdet from the grid has been chosen as an example. This results in 5185 examples for each seeder. Note that it is a common case for an example to have multiple clusterings with the same plagdet. This is because of the convex nature of the optimal setting described in Section 4.2.1. If this was the case for one example, the medoid setting has been chosen as the best. The medoid has been calculated using the Euclidean distance with epsilon and minPts being considered as the two coordinates in the Euclidean space of a parameter setting. For this calculation, minPts has been upscaled by a factor of 100. Otherwise, any difference in epsilon would completely neglect minPts.

To learn from these examples, the aforementioned distribution of the length of the seeds has to be encoded into features to train the model on. There are two possibilities for this encoding. First, to use statistics of the seeds like the total number or the mean, mode, or median length of the seeds. Second, to generate a histogram with fixed value ranges. The histogram has been chosen because it describes the distribution in more detail. The histogram features for a pair of documents are created by first rounding the length of all seeds down to their next multiple of 10 and then counting the number of seeds with a length of 0,10,...,290, and 300 and more. This results in 31 discrete features.

As for the model, a multilayer perceptron regressor (MLPR) was chosen. Using a neural network allows to easily verify certain properties of the relationship between the histogram and the parameters without fundamentally changing the underlying model. For example, if prediction performance of a three hidden-layer MLPR is similar to that of the same model with just one computing layer, then the parameters could be predicted independently. This does not mean the parameters are independent. Also, if the performance is comparable when changing the activation function from a linear to a non-linear one, then the function learned is linear in nature. For the experiments, the

MLPR implementation from scikit-learn[1] was used. For each experiment, the weights were initialized in a fixed random state. Concluding, the MLPR used has 31 input nodes, 31 hidden nodes per layer and two output nodes, one for epsilon and one for minPts. The scikit implementation does no computation in the input layer, so the number of hidden layers equals the number of computing layers. The MLPR used later in this chapter to predict the parameters has three hidden layers and uses a rectified linear unit (relu) as the activation function. In a prior test, there seems to be no notable difference between three or more layers. Also, relus provided more reliable results than sigmoid or htan neurons. Some experiments in Section 4.4 examine the choice of activation function and the number of layers in hindsight of the linearity of the function.

4.3.2 Evaluation Scheme

The evaluation scheme needs to show how close the predicted parameters are to the actual optimal ones. For this, plagdet as external metric and mean squared error as internal metric have been chosen.

Three plagdet measurements have to be considered for the external evaluation: the predicted document-level, collection-level, and optimal document-level plagdet. The predicted document-level plagdet represents the performance of an extender using the previously learned model to predict the DBScan hyperparameters for each seeder and each document based on the seeds. The model will be trained on examples from the `PAN13-Training` corpus, the plagdet will be calculated based on the predictions for the `PAN13-Test` corpus. The collection-level plagdet measures aligner performance when a static set of parameters has been used for every document. This static set will be determined via grid search on `PAN13-Training` and measured on `PAN13-Test`. For the optimal document-level pladget, a grid search has been done for each document pair in `PAN13-Test`. Then, the parameter set with the highest plagdet has been chosen for this document pair and those plagdets have been averaged for the whole collection.

Note that it can be expected that multiple parameter settings produce the same clustering, as described earlier. This means that the results of a plagdet comparison may equalize performance differences. Therefore, the mean squared error (MSE) will be, additionally, used as the internal metric to judge prediction quality in more detail. For internal performance evaluation, epsilon and minPts will be evaluated separately. In the training data, epsilon and minPts are extracted from the grid-points, so epsilon is between 0 and 2000 and minPts is between 2 and 9. Assuming the model will mostly learn this hypercube as a boundary for the parameters, the worst MSE on random

[1]For the experiments, Python 3.5 and scikit-learn 0.19 was used.

Table 4.2: Overview of the internal and external extension performance of different parameter prediction models. This table shows, how well the extension performs if the seeder is the same for training and test. The metrics are explained in Section 4.3.2.

Seeder		**MSE**		**Plagdet**		
learned	predicted	epsilon	minPts	pred. doc	coll-level	doc-level
$\varphi_{7,1}$	$\varphi_{7,1}$	69374	1.5	0.56	0.62	0.72
$\varphi_{7,2}$	$\varphi_{7,2}$	68620	2.7	0.63	0.75	0.87
$\varphi_{7,3}$	$\varphi_{7,3}$	92334	1.9	0.55	0.54	0.72
$\varphi_{7,4}$	$\varphi_{7,4}$	47822	2.6	0.47	0.57	0.67

predictions would be about $1000^2 = 1 * 10^6$ for epsilon and $3.5^2 = 12.5$ for minPts.

4.4 Results and Discussion

To recap the previous argumentation, predicting the parameters based on the seeds has some major advantages:

- While parameter estimation via grid search or manual tuning yields good results, it has to be done for each seeder and each combination of seeders. This is acceptable if only a few of those seeders are used. For large scale analyses, like automatic optimization or derivation of a seed-and-extend algorithm, those strategies quickly become infeasible.
- The optimal parameters vary between document pairs, so deriving static epsilon and minPts for one seeder is imprecise most of the time. Estimating the parameters for each document pair has the potential to increase extension performance by about 10%. This can be seen by comparing the collection-level plagdet with the optimal document-level plagdet from the experiment results show in Table 4.2, Table 4.3 and Table 4.4.

The experiments in this section show that parameter prediction is applicable reasonably well in those circumstances. The experiments analyze predictive performance in three distinct situations. The first one is how well the optimal parameters for one single seeder can be learned. The second is how well a model generalizes to seeder combinations. The third is how a model generalizes to unseen seeders. If the optimal parameters follow a functional relationship to the histogram of the seeds length, as described in Section 4.2.3, then the model should generalize well to at least combinations and seeders with a similar seed-length-distribution.

Table 4.3: Overview of the internal and external evaluation of different models predicting parameters for extension. This table shows how trained models generalize to seed combinations. The union symbol ∪ represents a seeder combination.

Seeder		MSE		Plagdet		
learned	predicted	epsilon	minPts	pred. doc	coll-level	doc-level
$\varphi_{7,1}$, $\varphi_{7,2}$	$\varphi_{7,1} \cup \varphi_{7,2}$	109082	2.1	0.70	0.69	0.86
$\varphi_{7,1}$, $\varphi_{7,3}$	$\varphi_{7,1} \cup \varphi_{7,3}$	106834	1.3	0.58	0.63	0.75
$\varphi_{7,1}$, $\varphi_{7,2}$, $\varphi_{7,3}$	$\varphi_{7,1} \cup \varphi_{7,2} \cup \varphi_{7,3}$	131665	1.9	0.67	0.69	0.85

Table 4.2 shows the results of four experiments where a network as described in Section 4.3.1 was trained on one of the seeders described in Table 4.1 each. Training examples were generated on the `PAN13-Training` corpus and the model was used to predict parameters for document pairs from `PAN13-Test`, generated by the same seeder the model was trained on.

For all four experiments over four different seeders, the predicted document-level plagdet is about 10% below the alternative collection-level plagdet, where only the best average parameter setting was used for each document pair. The outlier is the stopword 3-gram seeder $\varphi_{7,3}$, where the plagdet for predicted and collection-level estimated parameters is about even. Also, predictions are about 20% lower than the theoretically optimal document-level plagdet. The internal metrics here give a hint of how hard the parameters are to predict for each seeder and how high the spread of parameters is for examples with similar seeds. The average error for epsilon-predictions is between 210 and 300, the average error for predicting minPts is 1.2-1.6. Compared to the size of the optimal area of parameters, as can be seen in Figure 4.5, Figure A.5, Figure A.6, and Figure A.7, these errors are acceptable.

It can be assumed that the difference in plagdet between prediction and collection-level will be similar for all seeders with this model. Thus, if only one known seeder is employed, a static set of parameters is preferable. However, selecting parameters on document-level still has the potential to become better than collection-level, if those parameters can be estimated better. Nonetheless, the scenario where the seeder is the same for training and test is not the primary focus for parameter learning. If there are only a few, a priory known seeders, then manual tuning or more specialized models are preferable anyway.

More interesting is the capability of the model to generalize to seeder combinations and to unseen seeders. If the model can generalize to combinations reasonably well, then the model only needs to be trained once, with examples from each seeder on its own. Four experiments were made to show how well the model can predict parameters for seeder combinations when trained on the component seeders. The results can be seen in Table 4.3. It is notable that

Table 4.4: Overview of the internal and external evaluation of different models predicting parameters for extension. This table shows how good a trained model generalizes to unseen seeders. The union symbol ∪ represents a seeder combination.

Seeder		MSE		Plagdet		
learned	predicted	epsilon	minPts	pred. doc	coll-level	doc-level
$\varphi_{7,1}$	$\varphi_{7,3}$	99296	1.9	0.54	0.54	0.72
$\varphi_{7,1}$	$\varphi_{7,2}$	96904	4.5	0.48	0.75	0.87
$\varphi_{7,1}$, $\varphi_{7,3}$, $\varphi_{7,4}$	$\varphi_{7,2}$	91281	4.2	0.46	0.75	0.87
$\varphi_{7,1}$, $\varphi_{7,2}$	$\varphi_{7,4}$	50395	2.5	0.47	0.57	0.67
$\varphi_{7,1}$, $\varphi_{7,3}$	$\varphi_{7,1} \cup \varphi_{7,2} \cup \varphi_{7,3}$	147903	1.4	0.65	0.69	0.85

the difference between predicted document-level plagdet and collection-level plagdet is lower for all combinations compared to the single-seeder experiments. The largest difference is 5% for the combination of the token 5-gram exact match seeder $\varphi_{7,1}$ and the stopword 3-gram seeder $\varphi_{7,3}$. For the combination of $\varphi_{7,1}$ and the tf-isf cosine-similarity seeder $\varphi_{7,2}$, the prediction even performs marginally better than the static estimate via grid search. The difference between predicted and optimal document-level plagdet is with on average 17% (worst 18%) also slightly lower than for single seeders with 19% (worst 24%). Also, observe how the mean squared error for the epsilon prediction is higher than for the previous experiments, but the plagdet with predicted parameters is not worse. The conclusion of these experiments is that parameters vary more in combined seeders. This explains both the larger MSE of epsilon and the reduced performance of collection-level estimate. While the collection-level estimated parameters are better for single seeders, predicted parameters for seeder combinations are quite usable.

Besides generalizing to combinations is the ability to generalize to unseen seeders. If the theory holds that parameters follow a functional relationship to the seed-length-distribution, then it is possible to learn this from one seeder and predict reasonable parameters for a different one. Explicitly, a model has to predict, if the seeder used for testing and the one for training generate seeds with similar histograms.

Table 4.4 shows the results of the experiments about generalization to unseen seeders. The first experiment shows the prediction performance when the learned seeder ($\varphi_{7,1}$) has a similar seed-length-distribution of the predicted seeder ($\varphi_{7,1}$). The theory predicts that this generalization has to work well. This is indeed what the results show in that predicted document-level plagdet is the same as the collection level plagdet, which is the one from the third experiment in Table 4.2. The score of 0.54 when predicting $\varphi_{7,3}$ is not much lower than when predicting $\varphi_{7,1}$. This is quite different when learning on $\varphi_{7,1}$ and predicting $\varphi_{7,2}$, which has a very different seed-length-distribution (com-

Table 4.5: Changes in prediction error for multiple seeders for different activation functions and numbers of hidden layers. The seeders are the same for training and test. Here, the internal metrics have been calculated with a 30-fold 70:30 random resampling and over the trainings examples. For these experiments (only), weights have been randomly initiated for each fold.

			MSE	
seeder	layers	activation	epsilon	minPts
$\varphi_{7,1}$	1	relu	70601	1.5
	3	relu	**69860**	**1.5**
	1	none	72739	1.5
	3	none	75009	1.6

			MSE	
seeder	layers	activation	epsilon	minPts
$\varphi_{7,2}$	1	relu	71844	3.1
	3	relu	**70183**	**2.6**
	1	none	71246	3.1
	3	none	71247	2.8

			MSE	
seeder	layers	activation	epsilon	minPts
$\varphi_{7,3}$	1	relu	95671	1.9
	3	relu	**93865**	**1.9**
	1	none	111483	2.0
	3	none	116734	2.1

			MSE	
seeder	layers	activation	epsilon	minPts
$\varphi_{7,4}$	1	relu	47922	2.6
	3	relu	**46464**	**2.5**
	1	none	48716	2.6
	3	none	47559	2.9

pare Figure 4.6). Here, the prediction in notably worse. However, this has to be expected, since the model can only learn the parameters given in the training examples. Also, observe how the prediction of $\varphi_{7,2}$ does not improve when adding examples from $\varphi_{7,3}$ and $\varphi_{7,4}$ to the training set.

Generalization to combinations does, however, show promising results. When training a model on $\varphi_{7,1}$, $\varphi_{7,2}$, and $\varphi_{7,3}$ and predicting the combination $\varphi_{7,2} \cup \varphi_{7,2} \cup \varphi_{7,3}$, the difference between prediction and collection-level estimate is only 2%. When removing examples from $\varphi_{7,2}$ from the training set, the prediction performance decreases by only another 2%.

Concluding, generalization to unseen seeders works acceptably if the seed-length-distribution is similar, supporting the theory above. If the training set covers the different seed-length-distributions used when predicting, the model also generalizes to variations of them. This is a requirement for the extension when considering a search in the relaxation-space, for example. Also, predicting combinations seem to be reliable, even if the model has not been trained on all seeders used.

The last experiments, shown in Table 4.5, concern the model calibration. One of the reasons a MLPR was chosen is that it is easy to show if the function learned is linear in nature and if the parameters are interdependent for prediction, without completely swapping the underlying model. If the prediction error (here the MSE) is the same for one as for three computing layers, then the parameters can be learned independently. This is advantageous because it allows to use different, possibly more appropriate models in the future and tune

them differently. If the model performs equally after removing the activation function, then the learned function is linear.

The four combinations for 1 or 3 layers and with and without the activation function has been tested for each of the four seeders in Table 4.1. For each seeder, the three-layer MLPR with activation function performed best for both parameters. However, the difference between layers is marginal and it has to be assumed that the parameters can be learned independently. The difference in activation function is notable enough to assume that the learned function is not completely linear. However, the difference is low enough that a better but purely linear model might also predict better.

4.5 Conclusion

This chapter discussed the extension step of seed-and-extend text-alignment algorithms. Several alternative approaches have been analyzed, concluding that a density-based clustering is the generally most useful one for extension. DBscan has been adapted to work with seeds by providing *boxdistance* as a distance function for seeds, so the regular point-based DBScan algorithm can be used for 2-dimensional spaces.

Afterwards, several alternatives to determine good parameters for the clustering have been discussed, concluding that local optimization on internal validity metrics can not work, since having only one cluster in a document pair is a common case. Both collection-level estimation and parameter learning have been discussed as alternative ways to determine parameters for different seeders. The final conclusion of this analysis is that it is possible to learn good parameters for seeder combination. A parameter prediction model can also generalize to unseen seeders. However, this requires good training examples. This makes a large scale search in the relaxation and seeder combination space computationally feasible. However, collection-level estimation still performs better if the seeder and the truth is known beforehand, at least with the machine learning model used in this thesis.

It should be noted that there are some more specific clustering algorithms which might also solve seed extension. The LSDBC algorithm by Biçici and Yuret [6], for example, starts with the most dense points and expands the cluster until the density drops below a threshold. The HDBSCAN algorithm by Campello et al. [7] only needs minPts as hyperparameter and therefore could reduce the problem. Alternatively, the graph-based approach of MajorClust [34] might work well for extension but has a usually much larger runtime and also depends on hyperparameters.

Chapter 5

Future Work

This thesis presented a detailed look at the seed-and-extend strategy for text-alignment. The focus was on analyzing the performance and properties of the seeding and extension steps. However, certain aspects are yet to be explored for a comprehensive view on the topic. This final chapter will introduce some of these aspects and explores how to continue research. The first and most important section discusses the combined extension of seeds from different seeders. This should allow the construction of specialized seeders whose combination might improve detection capabilities. The second section discusses possibilities to find the optimal seed-and-extend text-alignment algorithm. The last two chapters discuss extensions and amendments to the alignment model and implementation.

5.1 Seeder Combination

The combination of seeds from different seeders has been mentioned and used several times in this work, for example in Section 4.2.3 or Section 3.1. The idea is to not search the single best seeder but to search for multiple seeders, each excelling in detecting specific relations. Consider the *summarization* category, for example. None of the singular seeders discussed in Section 3.2.2 could detect summarization-based reuse reasonably well. This is a difficult category for most seeders because they have to match a passage in one document to multiple passages in the other one. When using several, specialized seeders, where each detects a different pair of passages, this could be solved.

However, combining sets of seeds will introduce an extension problem. Even if two seeders match some passages differently, they will still find a lot of common ones. This is due to the nature of text similarity. Consider the following example: If two strings of text are very similar in syntax and semantics, they will be found by each seeder specializing in detecting either. If the similarity

between passages is coincidental the extension algorithm would usually discard them as outliers. If multiple seeders produce similar false positives these may form clusters themselves (as artifacts) or may be included in the existing clusters. This means that one can not simply combine every possible seeder and hope to improve detection. In conclusion, it is necessary to find a good selection of seeders for combination to improve detection performance.

5.1.1 Optimization Strategy

Assuming that all seeders in the relaxation space are admissible, including the most and least relaxed ones, seeder combination becomes combinatorial in nature. Thus, a heuristic approach is needed to find the best combination. To plan this heuristic, consider the following properties: First, the combine-operation is transitive. Combining two seeders simply means merging the sets of seeds they produce, as described in Chapter 3. Therefore, the ordering does not matter if all seeders will be added to the combined set in the end. Secondly, following the same argumentation as for relaxation in Section 3.1.4, the recall does not decrease when adding more seeds. This means that a combination of seeders never detects less than each seeder on its own, thus the goal should be combining as many seeders as possible to increase detection. However, the combination also merges all false positives. As described in Chapter 4, the extender detects and removes many of these false positives as outliers or noise. The clustering algorithms used for extension are largely robust against noise as long as outliers and noise are somewhat uniformly distributed and do not form artifacts. A way to determine if a combination of two sets of seeds forms artifacts is to use Hopkins statistic [5] on the false positives. Hopkins statistic is a measure for clustering tendency. It is calculated by averaging the distances of each point to its closest neighbor in a uniformly randomly distributed sample of points.

The two properties discussed above hint at a greedy, linear time heuristic: Assume a Markov-property for the combine operation and merge a seeder to the current state if the false positives are not grouped. This means the heuristic would loop over all seeders and decide for each if it should be added to the current state. A seeder that would produce artifacts if added to the current state would be discarded and denoted as incompatible. This scheme does not consider the relaxation effect. A relaxed seeder always produces the same seeds as its specialized version. This means, they also produce the same false positives, so combining a seeder and its relaxation would make artifacts more likely. Therefore, a heuristic like that would always include the first seeder encountered and discard these seeders neighbors in the relaxation space, independent of performance.

One solution to this problem is to pre-evaluate all considered seeders and find all groups of incompatible seeders. Then, the heuristic could choose the best out of every group and combine only those. Assuming this incompatibility is a pairwise property, it would suffice to compare the seeders in quadratic time, so the heuristic would still be faster than an exhaustive solution. However, it is more likely that incompatibilities are not pairwise but can occur for any number of combinations. In the latter assumption, detecting incompatible groups is again combinatorial.

A possible alternative to pre-evaluation is to order the seeders, improving the greedy search. There are multiple possibilities to sort seeders. Consider, for example, sorting by cost-benefit estimation. This means, order the seeders descending by precision. This way, it should take more iterations until the first seeders have to be rejected, so more seeders can be included and recall should increase. Also, consider sorting descending by F1-score or recall. This way, the best seeders will be added first. In the best case, after sorting by F1, the rejections would not contribute much to the performance anyways.

5.2 Search for the best Algorithm

Another promising project based on the properties discussed in this thesis is to search for the best possible text-alignment algorithm. Assuming that extension is mostly solved and need not be worried about, this leaves strategic search in the relaxation space and a strategic combination of the found seeders.

A promising strategy that incorporates both, seeder-construction and combination, is the genetic algorithm[1]. A genetic algorithm builds a population of individuals, breeds and mutates them and then kills the least fit individuals. Each individual holds one encoded hypothesis as his genome. For text-alignment, these hypotheses are seeders, the mutate operation is a step in the neighborhood of the relaxation space, the breeding operation is the combination of the seeders of both individuals and the fitness function is likely plagdet.

5.3 Framework Extension

Lastly, there are many possibilities to improve on the technical side of things. This means, developing more components to construct seeders with, improving runtime performance and include the more powerful, real-numbered matching.

[1] Observe that genetic algorithms nicely fit the biological theme of this thesis.

The components used in this thesis are rather primitive and mostly based on prior work in the field. With exception of the synonym operators, they all require the reused passages to be at least somewhat identical. This should be sufficient for reuse in terms of plagiarism and copyright infringement but not for complete information reuse. Recent developments in natural language processing, however, allow similarity comparison independent from character identity. For example, semantic similarity can be calculated based on word embeddings using Word Mover's Distance [18], syntactic similarity can be captured using syntactic n-grams [32].

In terms of runtime performance, indexing structures for matching are necessary. Currently, exact and sorted-identity matchers are built on an index structure. This reduces the runtime of the alignment algorithm significantly since matching without indexing structure is quadratic in time. It is also possible to find good indexes for other matchers. Approximate matching, like the Jaccard matcher, can be built upon suffix trees. Spacial matchers like cosine similarity are a tad harder since spacial index structures like R-Trees do not work efficiently in high dimensions. Performance can also be optimized with regards to third-party libraries. The framework uses many large libraries, for example, Deeplearning4j[2] for frequency-vector calculation. This is quite convenient but introduces a lot of overhead, especially in terms of size and memory usage. For many of the used third-party libraries, a tailored custom implementation could save a lot in terms of memory consumption and possibly runtime, as well as allowing more control over certain aspects of behavior and configurability. Both, resource consumptions as well as runtime performance are fundamentally important for more complex seeders and for the automated optimization mentioned in Section 5.1 and Section 5.2.

5.4 Real-Valued Matching

It was noted in Section 3.1 that the matching matrix M_h is always binary in this thesis. This constraint allowed for uncomplicated seeder combination and flexible use of extension algorithms. It was also noted that this matching does not have to be binary but could also be real-valued in [0, 1]. Such a matrix could allow for a more sophisticated extension since it includes information about the strength of the match. Also, seeder combination could consider the strength of the overlapping seeders.

[2]Deeplearning4j can be found at https://deeplearning4j.org/

5.5 Conclusion

Concluding, this thesis introduced a theoretical model for seed-and-extend text-alignment algorithms and an evaluation methodology to compare and optimize seeders and extenders for generic similarity relations. The results and conclusions of this work lay the foundation, together with seeder combination, to attempt to solve text-alignment also for complex reuse-cases like summarization.

Appendix A

List of Figures

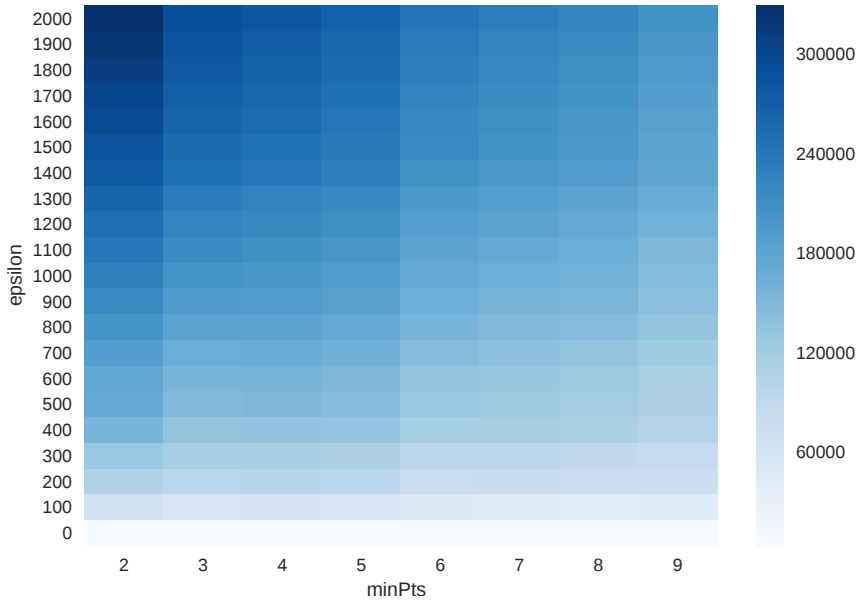

Figure A.1: Average Dunn index for different combinations of epsilon and minPts when defining the distance between clusters as Double.Max if there is only one or no cluster left. In this case the index increases drastically once the last clusters have been merged. In this case, increasing epsilon will include more noise points in the one cluster left, so this cluster grows and the Dunn index in this variation also increases. The optima of this function likely converges when epsilon becomes large enough to include all points in the same cluster. The seeder used is the same as in Figure 4.5.

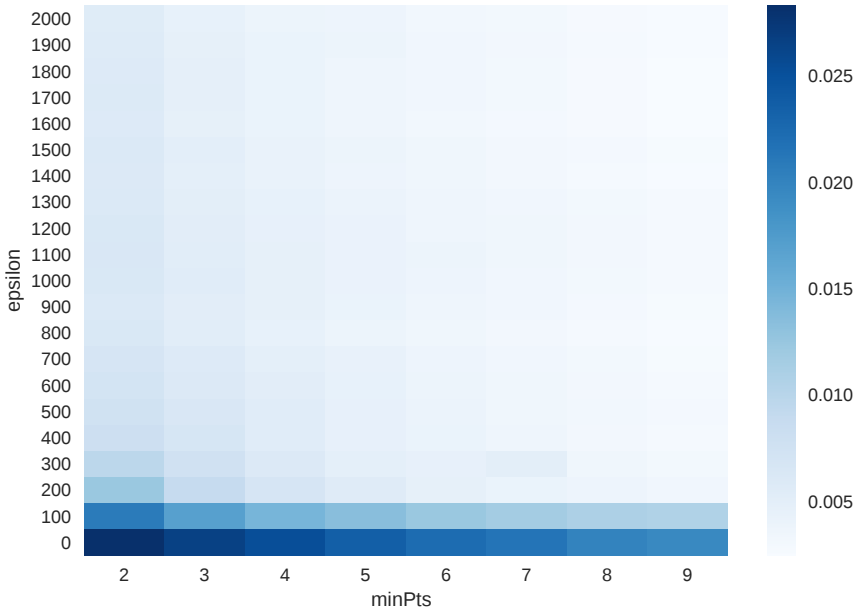

Figure A.2: Average Davies-Bouldin index (DB) for different combinations of epsilon and minPts. If there is only one cluster left, this DB variant becomes (close to) 0. The seeder used is the same as in Figure 4.5

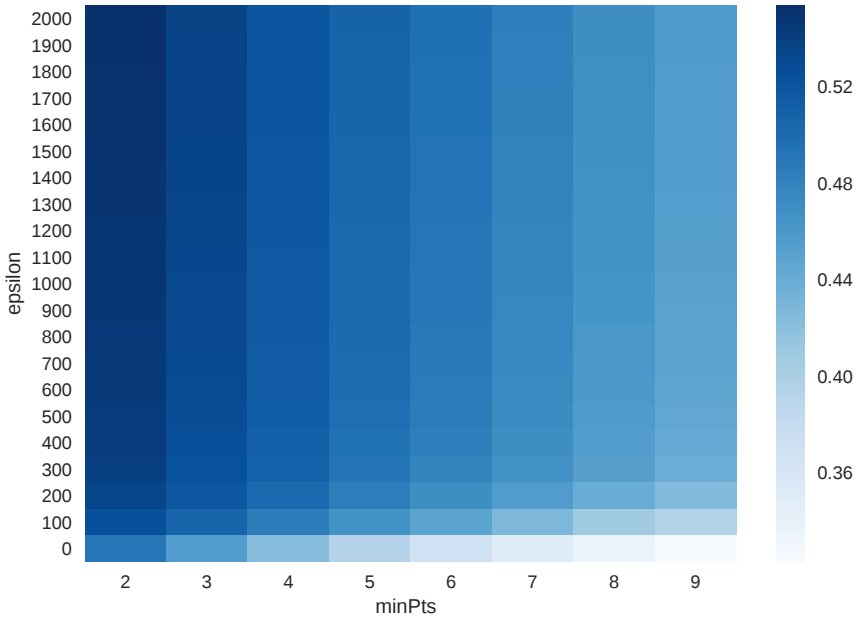

Figure A.3: Average silhouette for different combinations of epsilon and minPts. The seeder used is the same as in Figure 4.5

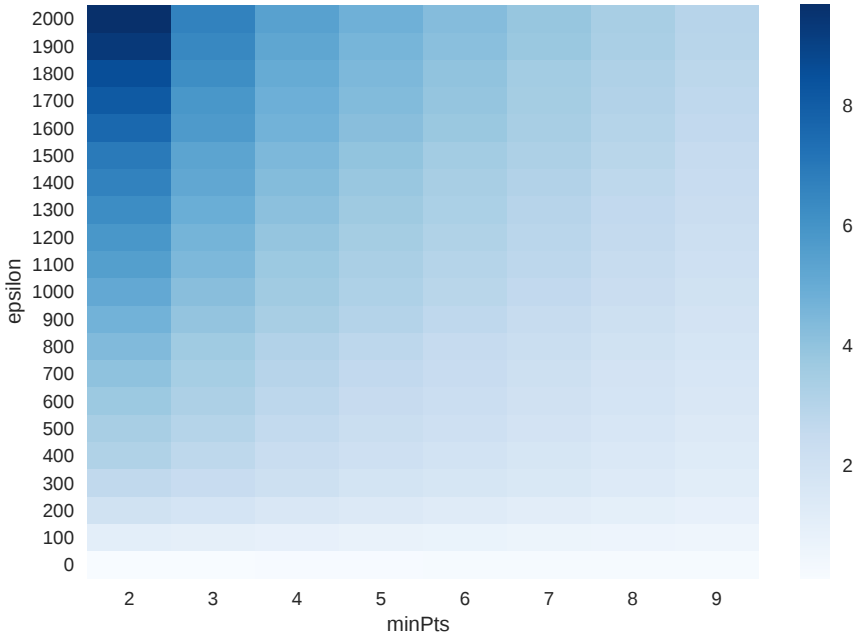

Figure A.4: Average scatter-to-length ratio for each parameter setting, where scatter-to-length ratio is defined as $\frac{1}{N}\sum_{i=1}^{N}\frac{s(C_i)}{\Delta C_i}$. As with other metrics that ignore the existence of multiple clusters, this measure is of linear nature to the length. A heuristic based on the scatter-to-length ratio would always merge more noise-points when given the chance.

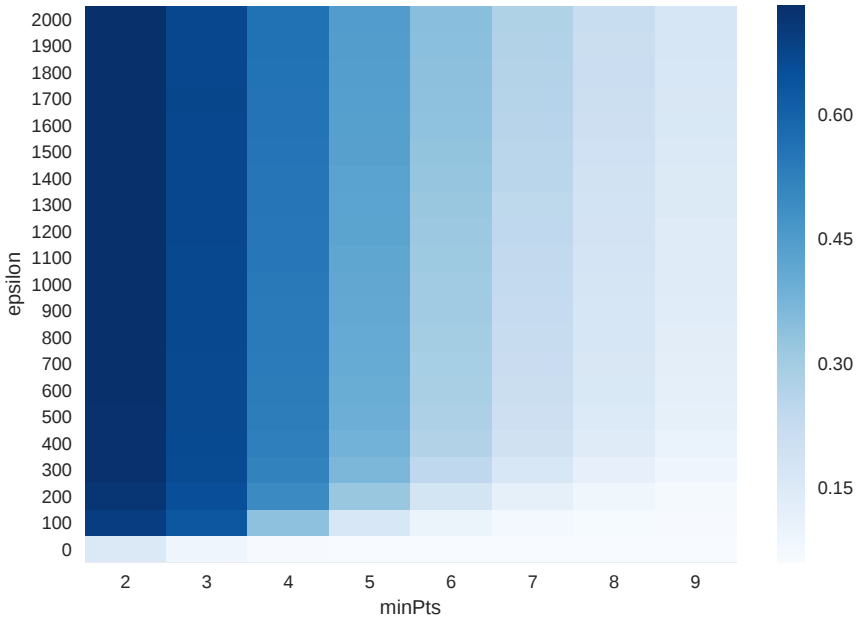

Figure A.5: Plagdet of DBScan extension over a grid of parameter choices. The static seeder used sentence unigram tf-isf vectors and cosine matching with a threshold of 0.33 (seeder $\varphi_{7,2}$ in Table 4.1).

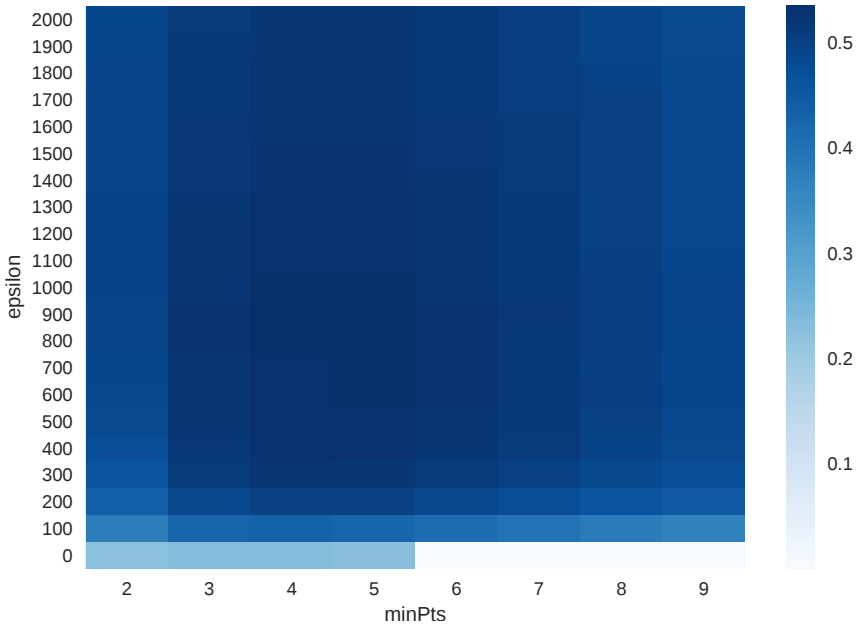

Figure A.6: Plagdet of DBScan extension over a grid of parameter choices. The static seeder used 2-overlap-3-grams of word tokens, exact matching and a negated stop-word filter (seeder $\varphi_{7,3}$ in Table 4.1).

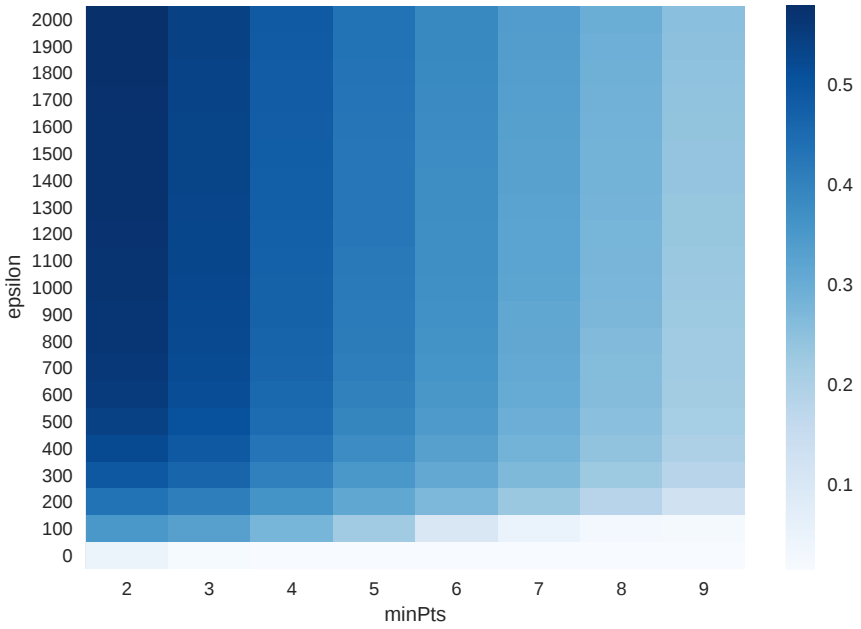

Figure A.7: Plagdet of DBScan extension over a grid of parameter choices. The static seeder used 2-overlap-3-grams of word tokens, exact matching and a negated stop-word filter (seeder $\varphi_{7,4}$ in Table 4.1).

Bibliography

[1] Samira Abnar, Mostafa Dehghani, Hamed Zamani, and Azadeh Shakery. Expanded N-Grams for Semantic Text Alignment—Notebook for PAN at CLEF 2014. In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *CLEF 2014 Evaluation Labs and Workshop – Working Notes Papers, 15-18 September, Sheffield, UK*. CEUR-WS.org, September 2014. 2.1, 2.1, 4.1

[2] AITools. URL `https://www.uni-weimar.de/de/medien/professuren/medieninformatik/webis/research/contributions-by-field/aitools/`. 3.2.1

[3] Faisal Alvi, Mark Stevenson, and Paul Clough. Hashing and Merging Heuristics for Text Reuse Detection—Notebook for PAN at CLEF 2014. In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *CLEF 2014 Evaluation Labs and Workshop – Working Notes Papers, 15-18 September, Sheffield, UK*. CEUR-WS.org, September 2014. 2.1, 2.1, 2.2, 2.3, 4.1.1

[4] Renato Cordeiro Amorim and Christian Hennig. Recovering the number of clusters in data sets with noise features using feature rescaling factors. *CoRR*, abs/1602.06989, 2016. URL `http://arxiv.org/abs/1602.06989`. 4.2.2

[5] Amit Banerjee and Rajesh N. Davé. Validating clusters using the hopkins statistic. In *IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2004, Budapest, Hungary, July 25-29, 2004.*, pages 149–153. IEEE, 2004. doi: 10.1109/FUZZY.2004.1375706. URL `https://doi.org/10.1109/FUZZY.2004.1375706`. 5.1.1

[6] Ergun Biçici and Deniz Yuret. Locally scaled density based clustering. In Bartlomiej Beliczynski, Andrzej Dzielinski, Marcin Iwanowski, and Bernardete Ribeiro, editors, *Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Warsaw,*

Poland, April 11-14, 2007, Proceedings, Part I, volume 4431 of *Lecture Notes in Computer Science*, pages 739–748. Springer, 2007. doi: 10.1007/978-3-540-71618-1_82. URL https://doi.org/10.1007/978-3-540-71618-1_82. 4.5

[7] Ricardo J. G. B. Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining, 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, April 14-17, 2013, Proceedings, Part II*, volume 7819 of *Lecture Notes in Computer Science*, pages 160–172. Springer, 2013. doi: 10.1007/978-3-642-37456-2_14. URL https://doi.org/10.1007/978-3-642-37456-2_14. 4.5

[8] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2): 224–227, April 1979. ISSN 0162-8828. doi: 10.1109/TPAMI.1979.4766909. 4.2.2

[9] J. C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974. doi: 10.1080/01969727408546059. URL https://doi.org/10.1080/01969727408546059. 4.2.2

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231. AAAI Press, 1996. URL http://www.aaai.org/Library/KDD/1996/kdd96-037.php. 4, 4.1.2

[11] Mark A. Finlayson. Java libraries for accessing the princeton wordnet: Comparison and evaluation. In Heili Orav, Christiane Fellbaum, and Piek Vossen, editors, *Proceedings of the Seventh Global Wordnet Conference, GWC 2014, Tartu, Estonia, January 25-29, 2014*, pages 78–85. University of Tartu Press, 2014. URL https://aclanthology.info/papers/W14-0111/w14-0111. 3.2.1

[12] Lee Gillam and Scott Notley. Evaluating Robustness for 'IPCRESS': Surrey's Text Alignment for Plagiarism Detection—Notebook for PAN at CLEF 2014. In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *CLEF 2014 Evaluation Labs and Workshop – Working*

Notes Papers, 15-18 September, Sheffield, UK. CEUR-WS.org, September 2014. 2.1

[13] Demetrios Glinos. A Hybrid Architecture for Plagiarism Detection—Notebook for PAN at CLEF 2014. In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *CLEF 2014 Evaluation Labs and Workshop – Working Notes Papers, 15-18 September, Sheffield, UK*. CEUR-WS.org, September 2014. 2.1, 2.1

[14] Philipp Gross and Pashutan Modaresi. Plagiarism Alignment Detection by Merging Context Seeds—Notebook for PAN at CLEF 2014. In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *CLEF 2014 Evaluation Labs and Workshop – Working Notes Papers, 15-18 September, Sheffield, UK*. CEUR-WS.org, September 2014. 2.1, 2.1

[15] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. EBL-Schweitzer. Cambridge University Press, 1997. ISBN 9780521585194. 2.1

[16] Apache Hadoop. URL `https://hadoop.apache.org/`. 5

[17] Leilei Kong, Yong Han, Zhongyuan Han, Haihao Yu, Qibo Wang, Tinglei Zhang, and Haoliang Qi. Source Retrieval Based on Learning to Rank and Text Alignment Based on Plagiarism Type Recognition for Plagiarism Detection—Notebook for PAN at CLEF 2014. In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *CLEF 2014 Evaluation Labs and Workshop – Working Notes Papers, 15-18 September, Sheffield, UK*. CEUR-WS.org, September 2014. 2.1, 2.1

[18] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 957–966. JMLR.org, 2015. URL `http://jmlr.org/proceedings/papers/v37/kusnerb15.html`. 5.3

[19] James Lewis, Stephan Ossowski, Justin Hicks, Mounir Errami, and Harold R. Garner. Text similarity: an alternative way to search medline. *Bioinformatics*, 2006. 2.1

[20] D.W. Mount. *Bioinformatics: Sequence and Genome Analysis*. E-libro. Cold Spring Harbor Laboratory Press, 2001. ISBN 9780879695972. 2.1

[21] Gabriel Oberreuter and Andreas Eiselt. Submission to the 6th International Competition on Plagiarism Detection. http://www.uni-weimar.de/medien/webis/events/pan-14, 2014. From Innovand.io, Chile. 2.1

[22] Yurii Palkovskii and Alexei Belov. Using Hybrid Similarity Methods for Plagiarism Detection—Notebook for PAN at CLEF 2013. In Pamela Forner, Roberto Navigli, and Dan Tufis, editors, *CLEF 2013 Evaluation Labs and Workshop – Working Notes Papers, 23-26 September, Valencia, Spain*, September 2013. ISBN 978-88-904810-3-1. 2.1

[23] Martin Potthast. Picapica text reuse analysis tools, 2018. URL `http://www.picapica.org/`. 2.3

[24] Martin Potthast, Tim Gollub, Matthias Hagen, Martin Tippmann, Johannes Kiesel, Paolo Rosso, Efstathios Stamatatos, and Benno Stein. Overview of the 5th International Competition on Plagiarism Detection. In Pamela Forner, Roberto Navigli, and Dan Tufis, editors, *Working Notes Papers of the CLEF 2013 Evaluation Labs*, September 2013. ISBN 978-88-904810-3-1. URL `http://www.clef-initiative.eu/publication/working-notes`. 2.2

[25] Martin Potthast, Matthias Hagen, Anna Beyer, Matthias Busse, Martin Tippmann, Paolo Rosso, and Benno Stein. Overview of the 6th international competition on plagiarism detection. In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *Working Notes for CLEF 2014 Conference, Sheffield, UK, September 15-18, 2014.*, volume 1180 of *CEUR Workshop Proceedings*, pages 845–876. CEUR-WS.org, 2014. URL `http://ceur-ws.org/Vol-1180/CLEF2014wn-Pan-PotthastEt2014.pdf`. 2.1, 2.3, 2.3

[26] Diego Antonio Rodríguez Torrejón and José Manuel Martín Ramos. CoReMo 2.3 Plagiarism Detector Text Alignment Module—Notebook for PAN at CLEF 2014. In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *CLEF 2014 Evaluation Labs and Workshop – Working Notes Papers, 15-18 September, Sheffield, UK*. CEUR-WS.org, September 2014. 2.1, 2.1

[27] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7. 4.2.2

[28] Miguel A. Sanchez-Perez, Grigori Sidorov, and Alexander Gelbukh. A Winning Approach to Text Alignment for Text Reuse Detection at PAN 2014—Notebook for PAN at CLEF 2014. In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *CLEF 2014 Evaluation Labs and Workshop – Working Notes Papers, 15-18 September, Sheffield, UK*. CEUR-WS.org, September 2014. 2.1, 2.1, 3.2.2, 3.3, 4.1

[29] Helmut Schmidt. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of International Conference on New Methods in Language Processing, Manchester, UK*, 1994. URL `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger1.pdf`. 3.2.1

[30] Erich Schubert, Alexander Koos, Tobias Emrich, Andreas Züfle, Klaus Arthur Schmid, and Arthur Zimek. A framework for clustering uncertain data. *PVLDB*, 8(12):1976–1979, 2015. URL `http://www.vldb.org/pvldb/vol8/p1976-schubert.pdf`. 4.1.2

[31] Prasha Shrestha, Suraj Maharjan, and Thamar Solorio. Machine Translation Evaluation Metric for Text Alignment—Notebook for PAN at CLEF 2014. In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *CLEF 2014 Evaluation Labs and Workshop – Working Notes Papers, 15-18 September, Sheffield, UK*. CEUR-WS.org, September 2014. 2.1, 2.1

[32] Grigori Sidorov, Francisco Velasquez, Efstathios Stamatatos, Alexander F. Gelbukh, and Liliana Chanona-Hernández. Syntactic n-grams as machine learning features for natural language processing. *Expert Syst. Appl.*, 41(3):853–860, 2014. doi: 10.1016/j.eswa.2013.08.015. URL `https://doi.org/10.1016/j.eswa.2013.08.015`. 5.3

[33] Matthew G. Snover, Nitin Madnani, Bonnie J. Dorr, and Richard M. Schwartz. Ter-plus: paraphrase, semantic, and alignment enhancements to translation edit rate. *Machine Translation*, 23(2-3):117–127, 2009. doi: 10.1007/s10590-009-9062-9. URL `https://doi.org/10.1007/s10590-009-9062-9`. 2.1

[34] Benno Stein and Oliver Niggemann. In Peter Widmayer, Gabriele Neyer, and Stefan Eidenbenz, editors, *Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, pages 122–134, Berlin Heidelberg New York, June . Springer. ISBN 3-540-66731-8. 4.5

[35] Benno Stein, Moshe Koppel, and Efstathios Stamatatos. Plagiarism analysis, authorship identification, and near-duplicate detection pan'07. *SIGIR Forum*, 41(2):68–71, 2007. doi: 10.1145/1328964.1328976. URL `http://doi.acm.org/10.1145/1328964.1328976`. 2.1

[36] Snowball Stemmer. URL `http://snowballstem.org/`. 3.2.1

[37] David Tolpin. Texhyphenator-j. URL `http://www.davidashen.net/texhyphj.html`. 3.2.1

[38] TT4J. URL `http://reckart.github.io/tt4j`. 3.2.1

[39] Princeton University. "About WordNet." WordNet. Princeton University. https://wordnet.princeton.edu, 2010. 3.2.1