Applying Graph Algorithms to Text Segmentation

Bachelor-Thesis von Marko Martin
October 25, 2010

TECHNISCHE UNIVERSITÄT DARMSTADT

Ubiquitous Knowledge Processing

Applying Graph Algorithms to Text Segmentation

vorgelegte Bachelor-Thesis von Marko Martin

Supervisor: Prof. Dr. Iryna Gurevych
Coordinators: Dr. Torsten Zesch, Nicolai Erbs

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Zusammenfassung

Textsegmentierung ist ein wichtiges Hilfsmittel im Natural Language Processing (NLP). Anwendungen finden sich in der Textzusammenfassung, im Information Retrieval und beim Auflösen von Anaphern. Frühere Arbeiten zum Thema der Textsegmentierung konzentrierten sich vor allem auf lexikalische Kohäsion, um Texte in Segmente einzuteilen. Diese Arbeit präsentiert zwei Algorithmen, die zu diesem Zweck semantische Graphen analysieren. Das Ziel dabei ist, die Qualität der Textsegmentierung mit mehr semantischen Informationen zu steigern als es in bisherigen Arbeiten der Fall war.

Diese Arbeit verfolgt drei Hauptziele: Zunächst werden frühere Arbeiten zum Thema Textsegmentierung präsentiert, analysiert und in Kategorien eingeordnet, die die Methoden anzeigen, die zum Bestimmen von Segmentgrenzen angewendet werden. Im zweiten Schritt werden zwei neue Algorithmen vorgestellt, die auf semantischen Graphen basieren. *Cluster Blocks* versucht, eine Segmentierung anhand von thematischen Clustern des semantischen Graphen vorzunehmen. *Compact Segments* ermittelt eine optimale Segmentierung im Hinblick auf die zwei Kriterien Kompaktheit und Grenzstärke, die für jedes mögliche Segment eines Textes analysiert werden: Kompaktheit bezieht sich auf die Stärke der semantischen Beziehungen innerhalb des Segmentkandidaten, und die Grenzstärke zeigt an, wie schwach die semantischen Beziehungen zwischen Termen innerhalb und außerhalb des Kandidaten sind. Die abschließend durchgeführte Evaluation zeigt die Qualität der neuen Ansätze im Vergleich zu bestehenden Verfahren unter Verwendung der Evaluationsmetriken P_k und WindowDiff. Compact Segments kann dabei auf kleineren Dokumenten mit State-of-the-Art-Algorithmen konkurrieren. Vier verschiedene Evaluationscorpora wurden genutzt, von denen einer, ein Wikipedia-basierter Corpus, für diese Arbeit entworfen und generiert wurde.

Abstract

Text segmentation is an important aid in Natural Language Processing (NLP). Applications can be found in text summarization, information retrieval, and anaphora resolution. Previous works on text segmentation mainly focused on lexical cohesion to divide texts into segments. This thesis suggests two algorithms which analyze semantic graphs for that purpose. The goal of this idea is to enhance text segmentation with more semantic information than it has been done in other works.

Three main goals are pursued in this thesis: First, related work on the topic of text segmentation is presented, analyzed, and classified into categories which indicate the methods that are used to determine segment boundaries. Second, two new segmentation algorithms are presented which are based on semantic graphs. *Cluster Blocks* tries to find segments in the text according to topical clusters of the semantic graph. *Compact Segments* identifies an optimum segmentation with respect to the two criteria Compactness and Boundary Strength which are analyzed for every possible segment of a text: Compactness pertains to the strength of semantic relatedness values within the segment candidate, and Boundary Strength indicates the weakness of semantic relatedness values between terms within and outside the candidate. Finally, an evaluation reveals the quality of the new approaches compared to existing ones by use of the evaluation metrics P_k and WindowDiff. Compact Segments proves to be able to keep up with state-of-the-art algorithms for smaller documents. Four different evaluation corpora have been used one of which, a Wikipedia-based corpus, has been designed and generated for this thesis.

Contents

1 Introduction

This chapter motivates the interest in text segmentation and introduces the goals of this thesis. Finally, it provides an overview of the content of the thesis.

1.1 Motivation

Text segmentation is an increasingly important topic of Natural Language Processing (NLP). It has therefore gained in attendance in the past twenty years of research. Its importance is becoming obvious when thinking of situations like these:

- **Information Retrieval.** You are searching the internet for a very specific information and type a query to your preferred search engine. As a result, you get a list of about 50 documents. Most of them are not obviously irrelevant, though not presenting the information you are interested in at first sight. Therefore, you have to call each page and search it manually – a time-consuming task since some of the pages consist of long texts, maybe online versions of book chapters. Wouldn't it be helpful to have the search engine doing this work for you, presenting you exactly the paragraph which is relevant for your query, and thus, helping you in evaluating each result quickly? Sure, the engine lists some sentences containing one or two of your query terms, however, this is in most cases the wrong snippet of the result page or the snippet is too short in order to actually understand what it is about. And this is what text segmentation can do for you here: After the search engine has found some relevant pages for your query, it will not present some of its content quite indiscriminately. Instead, it will segment the result page into topically coherent passages and present you exactly this very passage which best matches your query, thereby relieving you from the necessity of opening and searching the page manually.

- **Text Summarization.** You are searching the internet for documents pertaining to a certain topic and being of a certain degree of variety, i.e., you don't want to have them to be too specific and are rather searching for "overview" documents. Many of the results the search engine returned for your query seem to be very specific, however, you have to scroll through the whole result documents in order to verify that they really do not fit your needs. This is somewhat laborious; thus, you decide to examine only the text summaries offered by the search engine for the results, but you often find them not very helpful, especially because some contain many details pertaining to the same topic which should, in your opinion, be summarized in a single sentence. This is another task text segmentation can help you with: When the search engine creates the summary, it may first segment the text into its topically different sections and afterwards build the summary in a way which only uses one sentence per topic – which you would be very thankful for, wishing to have a good overview of the document topics.

There are even more tasks where text segmentation can be applied. Concrete applications of text segmentation can be found in Section 2.4.

1.2 Goals

The goals of the thesis are:

1. to give an **overview of state-of-the-art approaches** of text segmentation,

2. to elaborate some **new graph-based approaches**,
3. to build a new **evaluation corpora based on Wikipedia**, and
4. to **evaluate the new approaches** based on the most frequently used evaluation metrics P_k and WindowDiff.

The main goal is the exploration of some new graph-based approaches of text segmentation. This includes the theoretical justification of these methods as well as their implementation based on the UIMA framework[1] and evaluation on several text corpora which contain gold standard text segments.

The newly implemented methods should exploit the semantical graphs of texts which indicate the strength of relationships between words. They are to be explained in detail, including preprocessing steps, applied graph algorithms, possible parameter configurations, and final gaining text segments.

For evaluation purposes, a new corpora is to be built, using the English Wikipedia[2] as source.

Furthermore, as there are currently no UIMA evaluation components available for text segmentation, these are to be implemented as well. They should use two measures of text segmentation evaluation, namely P_k and WindowDiff which have become kind of standard for evaluating segmentation algorithms, in order to quantify the performance of the implemented segmentation algorithms and make it comparable to state-of-the-art approaches. These measures will also be presented in detail to allow for a deep understanding of the measure values.

1.3 Structure

The thesis is mainly divided in an introductory part – the Introduction and Background Chapters – and the chapters after it, presenting the main work of this project. Thus, the work is structured as follows:

Chapter 1	Motivates the interest in text segmentation and introduces the goals of this work and its structure.
Chapter 2	Gives some definitions pertaining to text segmentation and provides a basic understanding of some important topics in Natural Language Processing. Introduces the most important recent text segmentation approaches. Presents some applications of text segmentation.
Chapter 3	Presents two new algorithms for text segmentation which are based on semantic graphs.
Chapter 4	Introduces the two most frequently applied metrics for evaluation of text segmentation algorithms. Presents corpora which have been used for evaluation. Clarifies the way in which the new algorithms have been tuned and the best configurations have been chosen. Finally, presents and analyzes results of evaluation.
Chapter 5	Dwells on implementation issues. Particularly, explains the pipelines and software components that have been used in the course of the thesis.
Chapter 6	Sums up the contents of the thesis. Draws some conclusions and gives an outlook to future work.

Lists of figures, tables, and references are located in the appendix.

[1] http://uima.apache.org/
[2] http://en.wikipedia.org/

2 Background

This chapter provides an overview of general concepts of Natural Language Processing which are relevant to the topic of text segmentation. Furthermore, it presents recent text segmentation approaches and lists applications of text segmentation.

2.1 General Concepts of Natural Language Processing

This section is meant to give a general introduction to the main concepts of Natural Language Processing (NLP) which are somehow related to the task of text segmentation. The explanations and definitions given in this section will be referred to in later sections and chapters. Therefore, we suggest reading this section before other parts of the thesis if the reader is not that familiar with NLP concepts in general or text segmentation concepts in particular.

This section is *not* meant to provide a coherent presentation of text segmentation methods. For a description of complete text segmentation methods, refer to Section 2.3.

2.1.1 Preprocessing

In any task of Natural Language Processing, generating features from one or more original texts is a crucial task since the way in which tokens and sentences are represented significantly influences the results of algorithm applications. This process of feature generation is therefore part of preprocessing of any NLP method.

Tokenization

Tokenization is the process of splitting a natural-language text into tokens, i.e., words that are to be considered separately. This process can be implemented differently in the following points (not complete):

- Tokens may or may not be split at apostrophes. E.g., the word “that’s” may be split into “that” and “s” (or “is” for a kind of intelligent splitting, respectively).
- Tokens may or may not be clustered to phrases where possible. E.g., the words “New York” may be considered as only one token in order to conform to its actual meaning.
- Tokens may or may not be converted to all-lower-case (or all-upper-case). Most algorithms profit from this conversion.
- Often, tokens are also merged into groups though not belonging to a phrase. These groups, called n-grams if consisting of exactly n tokens, help in reducing the number of features and in avoiding ambiguities since co-occurrents of a word often contribute to its contextual meaning.

Stemming

Stemming is a method which reduces words to their stems, aiming at reducing the number of different tokens in a text. E.g., it may transform the words “library” and “libraries” to the same stem “librari”, ameliorating the discovery of semantical equality of words. Stems are generated without usage of a dictionary; instead, this is tried to be performed with simple transformation rules, based on the language of the stemmer. The most popular stemmer implementation has been developed by Porter [Por97].

Lemmatization

Lemmatization finds the basic form of a word and, similarly to stemming, aims at reducing the number of different tokens in a text. However, lemmatization is even stronger than stemming since it usually also finds basic forms which cannot be obtained heuristically. E.g., the words "goes" and "went" might both be reduced to "go" whereas a stemmer would not reveal a common root of the words. Moreover, lemmas are natural basic forms whereas stems tend to be artificial (see the "librari" example in the previous section).

POS Tagging

Part-of-speech (POS) tagging is a method of generating features pertaining to the part of speech of words. This can help in resolving ambiguities where, for instance, a substantive may, morphologically, also be a verb. Many NLP algorithms also remove words from the set of features if they do not belong to the kinds of words which are assumed to be helpful for the particular task (often nouns, verbs, adjectives).

Stop words

Stop words are words which appear very frequently across all kinds of texts and thus have low semantic expressiveness. Such words (e.g., "yes", "no", "is", "can") are often removed from the set of tokens as they do not help in semantic analyzation of texts.

2.1.2 Similarity and Cohesion

Many NLP tasks considered with some kind of semantic analysis of texts rely on methods of measuring similarities of tokens, sentences, or even greater blocks of text. In many cases, also measures for the cohesion of a single block are applied which are designed to produce greater values if a block is more "semantically coherent". However, a unique definition of semantic cohesion is not existing.

This subsection lists some frequently applied concepts of similarity and cohesion.

Token Similarity

Different concepts exist for exploiting token similarities:

- *Morphological equality* is the most trivial concept and relies on the character sequence of tokens[1] only. However, as analyzed by Morris and Hirst [MH91], simple word reiterations may be an important hint to semantical relationships within a text.
- Word *co-occurrences* may be significant for discovering semantic relationships. If the contexts of two words have many words in common, this might be an indicator for the semantic relationship of those words. This idea is applied for text segmentation by Ferret [Fer07].

[1] Note that tokens are usually stemmed; thus, words such as "library" and "libraries" are considered to be morphologically equal.

- A *thesaurus* is a very powerful tool for finding semantic relationships. Most thesauri define relationships such as categories, synonyms, hypernyms, hyponyms etc. which allow for establishing links between different words that do not refer to each other by a similar context. Fellbaum, for instance, suggests *WordNet* for calculating relations between words [Fel98].

Sentence/Block Similarity

Particularly for text segmentation purposes, determining the similarity of whole sentences or blocks of texts is of high importance. Popular concepts of block similarity are the following:

- Number of *common features*, often relative to the number of all features
- The *Euclidean distance* considers the blocks to be compared as vectors which usually contain for each token of the text the number of its appearances in the respective blocks. The distance of the vectors u, v is then calculated as $d = \|u - v\| = \sqrt{\sum_i (u_i - v_i)^2}$.
- The *cosine similarity* also works on the block vectors. Instead of measuring the vector distances, it calculates the cosine of the angle between the vectors: $d = \frac{u \cdot v}{\|u\|\|v\|}$. In contrast to distance measures, this measure yields higher values (up to 1) for similar blocks.

Lexical Chains

Many algorithms based on lexical cohesion use *lexical chains* for structuring texts according to lexical similarities [MH91]: A lexical chain is a range of text which excels in a high lexical cohesion which means that many of the tokens contained in the chain are lexically related to each other (see the possible definitions of token similarities). E.g., if five consecutive sentences of a text contain names of states of the USA (which are obviously related), they will likely be combined in one lexical chain. One lexical chain usually pertains to exactly one semantic concept. Consequently, a text passage may be part of multiple lexical chains if it contains multiple semantic concepts.

Lexical chains have first been mentioned by Halliday and Hasan [HH76] who introduced the concept but did not provide a concrete algorithm for building the chains. For an example of such an algorithm, refer to Section 2.3.1.

2.1.3 Lexical-Semantic Graphs

In various NLP tasks, including text summarization, keyphrase extraction, and query answering, it can be useful to consider the lexical-semantic relatedness of arbitrary word pairs of a text. This, in fact, raises the need for building a complete graph consisting of all relatedness values of the contained words, the so-called *Lexical-Semantic Graph*: It consists of one vertex for each term and edges between all of them, weighted with a lexical-semantic relatedness value indicating the strength of similarity between the connected terms. These similarity values usually base on one or more of the categories presented in Section 2.1.2 and provide a numeric value of concept similarity.

A lexical-semantic graph may actually be based on different kinds of features. Lemmas and noun phrases have, in many applications, turned out to represent the semantic structure of a text most accurately. For example, the text presented in Figure 3.1 might look like Figure 3.3 in a sematic graph representation if lemmas are considered as features.

Zesch [Zes09] gives an extensive overview of existing and adapted relatedness measures and of possible resources to exploit for extracting semantic relations. In description of applications, he focuses on keyphrase extraction and presents a new method based on semantic graphs for discovering important terms of documents. Another keyphrase extraction algorithm based on semantic graphs has recently been proposed by Tsatsaronis et al. [TVN10].

A method for obtaining semantic representations of texts that has proved to yield good results for semantic relatedness measurement and that we will come back to later in the thesis is *Explicit Semantic Analysis (ESA)* which has been proposed by Gabrilovich and Markovitch [GM07]: They represent input texts (for which to calculate semantic relatedness to other texts) as concept vectors. A so-called semantic interpreter takes the text T as input (which may be only one word or even a whole paragraph) and iterates over the words. The word concept vectors are then weighted and added to obtain the text concept vector.

Word concept vectors are constructed from a big human knowledge repository such as Wikipedia: For every concept of the repository (e.g., for each article of Wikipedia), the vector contains one entry indicating the relevance of the concept for the word. This value is calculated using the TF-IDF[2] measure [SM86], i.e., if v is the concept vector of word w and v_j is the entry for concept no. j, it holds:

$$v_j = \frac{\text{freq}\left(w, T_j\right)}{\text{docfreq}(w)}$$

where T_j denotes the text belonging to concept j, freq gives the frequency of a word in a text, and docfreq gives the number of concept documents in which a word appears. (The TF-IDF value is often also normalized or adapted by taking the logarithm of the inverse document frequency.)

By merging all concept vectors of the words in a text, the ESA method generates a concept vector for the whole input text. Two such texts can then be compared using a vector comparison method applied to the concept vectors of the text, e.g., the cosine measure. This finally yields a value indicating the semantic relatedness of the input texts.

2.2 Text Segmentation

This section introduces the task of text segmentation. It presents possible categorizations of segmentation approaches and clarifies on which of the categories the focus of this thesis lies on. It furthermore lists cues of documents which may be exploited for text segmentation.

2.2.1 Definition

Text segmentation is a special case of topic segmentation only considering documents in written form. Topic segmentation is, according to Purver et al. [PGKT06], the *division of a text or discourse into topically coherent segments*.

2.2.2 Categorization

Segmentation algorithms can be categorized by means of different criteria:

[2] term frequency – inverse document frequency

- *Feature exploitation* according to [Yaa97]: Algorithms based on *lexical cohesion* exploit the fact that semantically related sentences consist of lexically similar tokens [HH76]. Algorithms based on *multiple sources* also try to discover other relationships of sentences and tokens beyond their lexical similarity. This may include syntactical or even prosodic cues if speech recording is available.
- *Linear/hierarchical*: Segmentation approaches may either produce a pure *linear* segmentation of a text, thus, simply placing boundaries between appropriate sentences of the text, or yield a *hierarchical* division of a text, thus, providing coarse-grained segments which are themselves divided into more fine-grained segments.
- *Learning strategy*: *Supervised* algorithms are trained on example texts before measuring their performance on other texts whereas *unsupervised* algorithms are not. The latter ones are usually mainly based on lexical cohesion and not on other features such as cue phrases (see below).

This work concentrates on approaches which

- exploit **lexical cohesion only**,
- are **linear**,
- and **unsupervised**.

2.2.3 Segment Cues

Diverse cues are appropriate for being exploited for text segmentation. Frequently used are the following:

- The most important cue is *lexical cohesion* which is also the most general one: Lexical cohesion may pertain to word reiterations, word categorizations, and co-occurrences among others as we have seen in Section 2.1.2. Lexical cohesion cures are always applicable in some way since, at least for simple cues such as word reiterations, no information is needed beyond the text. Lexical cohesion in general is often considered to be the most useful signal for finding text segments since coherent segments usually consist of lexically related words.
  According to Reynar [Rey98], lexical cohesion cues useful for text segmentation are:
  - *First uses*: If a word appears the first time, it may indicate a lexical break in the text.
  - *Word repetitions* indicate lexical cohesion.
  - *Word n-gram repetitions* indicate lexical cohesion with an even greater probability since n-grams are less likely to be repeated than single words.
  - *Frequency of a word* suggests high lexical cohesion of two blocks if the frequency is high in both blocks, relative to a priori knowledge of average word frequencies. (E.g., frequent occurrence of the word “are” in two neighbored blocks is not a strong indicator of cohesion since “are” is very frequent across all texts.)
  - *Synonymy* of words indicates lexical cohesion.
- *Cue phrases* are frequently an indicator of segment changes, too. Phrases like “Now, ...”, “... is a/an ...”, “In contrast ...” might suggest a topic change while “Furthermore, ...” or “On the other hand, ...” might signal a continuation of a train of thought within a segment. However, exploitation of cue phrases is usually difficult in an unsupervised approach since they substantially differ between different kinds of texts.
- *Intonational, prosodic, and acoustic cues* can be helpful if a speech recording is available because segment boundaries will likely tend to take place at positions of longer breaks or speaker changes, for instance [GMFLJ03].

2.3 Related Work

This section introduces the most important approaches and ideas of recent works exploiting lexical cohesion for text segmentation. Figure 2.1 provides an overview where approaches are arranged according to their time of publication and to the main concepts they are based on:

- **Lexical Scores**: This category contains all approaches which explicitly calculate scores for the cohesion of blocks (sequences of sentences) and/or the similarity of blocks for deriving segment boundaries from the results.
- **Lexical Chains**: This category holds works which build lexical chains for deriving segment boundaries from the chain positions in the text.
- **Clusters/Partitions**: These works apply a clustering or partitioning algorithm to obtain the segments.
- **Probabilistic**: These algorithms introduce probabilistic models according to which the optimum segmentation is calculated.

2.3.1 Construction of Lexical Chains

As many works are based on lexical chains, we will first look at their origins: The first algorithm to create lexical chains was proposed by Morris and Hirst [MH91]. In constructing the chains, the authors use a thesaurus for determining semantic relationships such as category or group equality between tokens. Their algorithm iterates over all tokens of the text and assigns each token to a lexical chain, either a new one – if no chain for the word category is existing –, or an existing one – if a chain for the word category is existing and adding the word would not make the chain exhibit a gap of more than three sentences without a word belonging to the chain. Most segmentation algorithms using lexical chains construct them this way or at least similarly. The first linear-time algorithm for construction of lexical chains was proposed by Silber and McKoy [SM02].

2.3.2 Lexical Chains

Okumura and Honda (1994)

One of the earliest algorithms for exploiting lexical chains for text segmentation was proposed by Okumura and Honda [OH94]. The authors' main idea is that lexical chains should conform to the structure of a text, thus allowing for a segmentation by finding places in the text where the density of chain start and end points is particularly high.

The algorithm first builds the chains putting words to the same chain if their categories in the used thesaurus are equal. If no such chain exists for a word, a new chain is created. If a word has several categories (i.e., if a word's meaning is ambiguous), the word is added to the most salient chain of those which pertain to one of the word's categories. A chain is considered to be more salient than another one if it is longer or if a word belonging to the chain has appeared more recently (in a lower distance from the currently analyzed word) than a word belonging to the other one. This strategy is a kind of implicit word sense disambiguation.

The next step is calculating the "boundary strength", the number of beginning or ending lexical chains at each sentence gap. These values are then sorted in descending order, and the more start and end points of lexical chains are present at a sentence boundary, the more likely this boundary will also be chosen as a segment boundary. Okumura and Honda illustrate this idea as in Figure 2.2: Lexical chains tend to begin and end at certain points (here: before sentence 14), indicating segment boundaries.

Galley et al.: LCseg (2003)

Galley et al. propose *LCseg* [GMFLJ03], another popular segmentation algorithm. Although the authors claim having used lexical chains, in fact, the used chains only pertain to term repetitions and are therefore a different concept than what had originally been defined as lexical chains by Morris and Hirst [MH91]. (Nonetheless, we will call the chains "lexical" to conform to the authors.)

Each lexical chain pertains to a certain (stemmed) token, and if this token is found again during chain construction, it will be added to its chain. Chains are split if they contain long ranges (exceeding a certain length which is a parameter of the approach) without any appearance of its associated token.

The algorithm then assigns weights to lexical chains where chains receive higher scores than others if they contain more term repetitions or if they are shorter. Similarly to TextTiling (Section 2.3.3), lexical scores are assigned to sentence breaks indicating the strength of cohesion of neighboring blocks.

The score for a gap between two blocks A and B is calculated using the cosine measure applied to vectors containing the weights of lexical chains overlapping block A and B, respectively. Vector entries for lexical chains which do not overlap the particular block are zero. E.g., if weights for the lexical chains $1, 2, 3$ of a text are $w_1 = 10, w_2 = 15, w_3 = 17$, and chains 1 and 2 overlap block A, 2 and 3 overlap block B, the block vectors would be:

$$v_A = \begin{pmatrix} 10 & 15 & 0 \end{pmatrix}, v_B = \begin{pmatrix} 0 & 15 & 17 \end{pmatrix}$$

The lexical score assigned to the gap between blocks A and B would in this case be:

$$s_{AB} = \frac{v_A v_B}{\|v_A\| \, \|v_B\|} \approx 0.55$$

Gaps with low scores that constitute sharp minima of the score function are chosen as segment boundaries. The procedure is the same as applied by TextTiling (see Section 2.3.3).

Marathe and Hirst (2010)

The work of Marathe and Hirst [MH10] stems from Okumura's and Honda's approach: Both first build lexical chains and then count the number of beginning and ending chains at each sentence gap. These values, the boundary strengths, are finally sorted in descending order to obtain boundaries which are appropriate to become segment boundaries.

The difference of both approaches is the way in which lexical chains are built: While Okumura and Honda use thesaurus categories to represent the concepts of lexical chains, Marathe and Hirst construct lexical chains using token-based semantic similarity measures: The first used measure is Lin's WordNet-based measure [Lin98], the second one is Mohammad's and Hirst's framework of distributional measures of concept distance [MH06].

For each token of the text, the chain with the highest similarity according to the similarity measure is chosen to be extended by that token if the calculated similarity value exceeds a fixed threshold. If

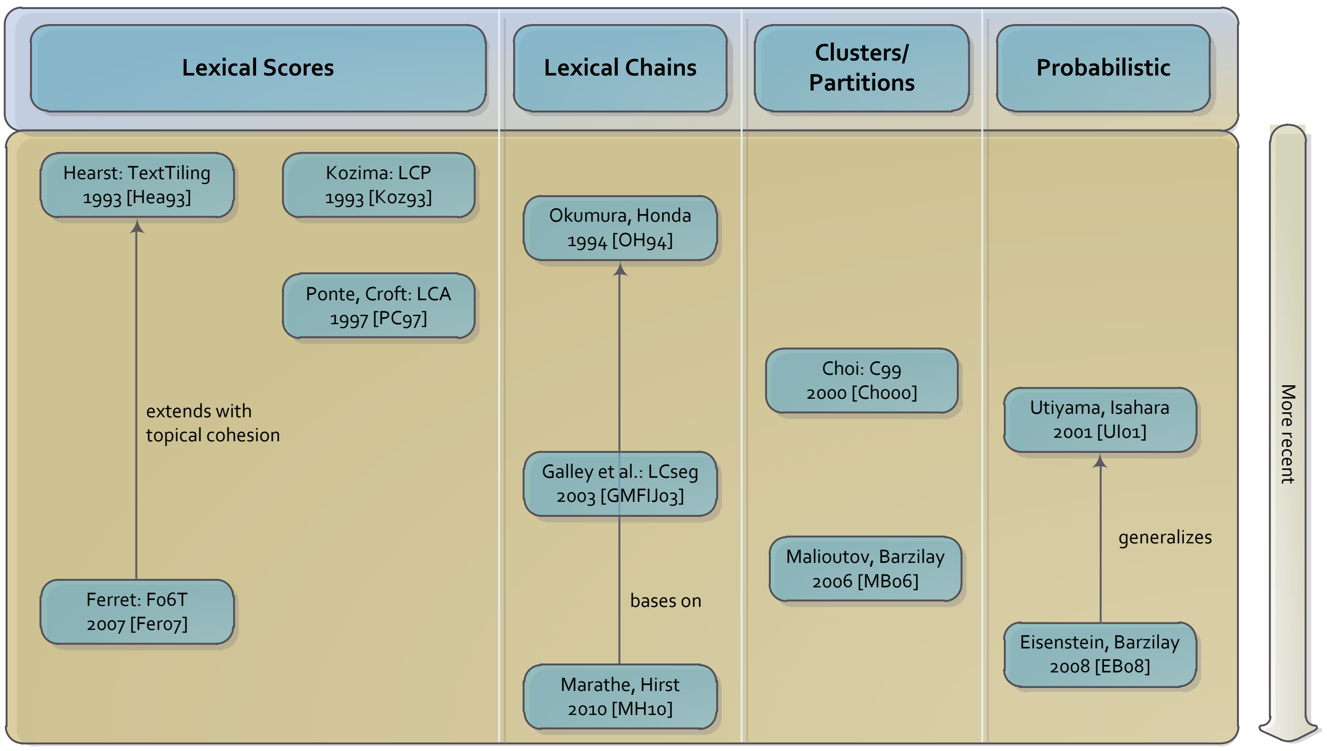

Lexical Scores
Lexical Chains
Clusters/ Partitions
Probabilistic
Hearst: TextTiling 1993 [Hea93]
Kozima: LCP 1993 [Koz93]
Ponte, Croft: LCA 1997 [PC97]
extends with topical cohesion
Ferret: Fo6T 2007 [Fer07]
Okumura, Honda 1994 [OH94]
Galley et al.: LCseg 2003 [GMFLJ03]
bases on
Marathe, Hirst 2010 [MH10]
Choi: C99 2000 [Cho00]
Malioutov, Barzilay 2006 [MB06]
Utiyama, Isahara 2001 [UI01]
generalizes
Eisenstein, Barzilay 2008 [EB08]
More recent

Figure 2.1: Overview of important text segmentation approaches

```
chains    | text
          |          1         2
start-end | 123456789012345678901234
( 1 - 24) | ************************
( 4 - 13) |    **********
(14 - 16) |              ***
( 8 -  9) |        **
(14 - 18) |              *****
```

Figure 2.2: Lexical chains illustrating the idea of boundary strengths (adopted from Okumura and Honda)

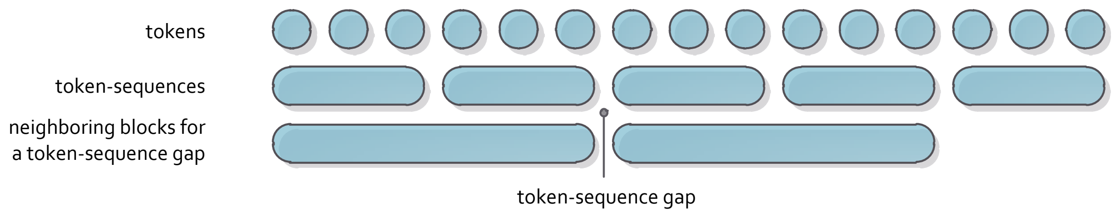


Figure 2.3: Tokenization in TextTiling

multiple chains are found to have a sufficiently high similarity, these chains are merged if their pairwise similarity is high enough. If there is no appropriate chain for a token, a new chain is created for it.

For a small corpus of 20 documents and using Mohammad's and Hirst's measure of concept distance, this approach achieves better results, using the WindowDiff metric (see Section 4.2.2), than TextTiling (Section 2.3.3) and C99 (Section 2.3.4).

2.3.3 Lexical Scores

Hearst: TextTiling (1993)

Besides LCseg, there are even more famous algorithms which do not take into account the full range of lexical cohesion, but only morphological equality of tokens. E.g., with *TextTiling* ([Hea93], [Hea94], [Hea97]), Hearst presents an algorithm which abandons lexical chains as she states having found them to be inappropriate for text segmentation purposes, particularly due to word ambiguities that would make tokens to be assigned to the same chain although they have actually a different meaning in the context of the text. Instead, the author does not rely on lexical similarity, but calculates token similarity according to their morphological similarity only.

The text to be segmented is first divided into artificial token-sequences (n-grams) in order to avoid having units of different size. Lexical scores are then calculated for every token-sequence gap, expressing the similarity of the neighboring blocks (see Figure 2.3). Block similarity is, in the default configuration, determined with the cosine measure applied to the word frequency vectors of the blocks.[3]

Based on this, so-called depth scores are finally calculated for each token-sequence gap in order to determine the strength of change in subtopic cues in the neighboring blocks. For this purpose, the block similarity values are first smoothed with a simple averaging method. Then, depth scores are obtained in the following manner for each token-sequence gap: Starting at the current gap, the algorithm moves left as long as similarity values are increasing. The difference of the last (thus, greatest) of these values and the similarity value of the initial gap denotes the "left" depth. The same procedure is repeated for the right site of the gap. The final depth score is calculated by adding "left" and "right" depth values.

The list of depth scores, sorted in descending order, is used to obtain segment boundaries. The number of segments is determined by a cutoff value for depth scores to be included. This cutoff value S depends on the average $\bar{s}$ and the standard deviation σ of depth scores and is usually implemented as $S = \bar{s} - \frac{\sigma}{2}$.

[3] An alternative configuration measures similarity by the extent of new vocabulary introduced in the blocks.

Ferret: F06T (2007)

Ferret follows a similar idea for calculating depth scores. However, he extends the calculation of cohesion values by taking into account topical relationships between textual units. Thereby, he abandons usage of external knowledge [Fer07].

Thus, the first step of his algorithm *F06T* is identifying the topics covered by a text. For this purpose, co-occurrences are examined for each word of the text, assuming that frequently appearing co-occurrents of a word are topically related to it. As a result, a similarity matrix of words is constructed where similarity values are calculated using the cosine measure applied to the co-occurrent frequency vectors of words.

The matrix is then converted to an initially complete graph of the words. The Shared Nearest Neighbor algorithm [ESK01] is applied to obtain a clustering of this graph where the clusters constitute topically related words. Clusters are referred to as "topics".

Afterwards, each sentence gap is assigned a cohesion value with respect to the words within a fixed-size window centered at the gap. Two properties effect this value:

1. The ratio between words appearing on both sides of the window and all words within the window. (This is the pure morphological contribution, here used instead of the cosine similarity which is applied, e.g., in the *TextTiling* approach.)

2. The ratio between words which are topically expressive for the window and all words within the window. For this calculation, first, the topics associated with the window are determined by calculating the cosine similarity of each topic vector (consisting of 1-entries for each word in the topic cluster according to the clustering gained from the previous step) and the word frequency vector of the window. If similarity exceeds some fixed threshold, the topic is assumed to be relevant for the window. Topically expressive words are then defined as those belonging to topics which have been marked relevant for the window.

Both calculated ratio values are then added to obtain the global cohesion value, respecting both morphological and topical cohesion. Like with *LCseg* and *TextTiling*, sharp minima of the cohesion value function are finally chosen as segment boundaries.

Kozima: Lexical Cohesion Profile (1993)

One of the earliest ideas for segmenting texts based on its lexical cohesion has been documented by Kozima [Koz93]. Instead of building lexical chains, the author's algorithm moves a fixed-sized window over the text, stopping at each token, and for each position measures the lexical cohesion of the range covered by the window.

The cohesion of a range is calculated according to a method exploiting semantic relationships of tokens contained in that range (see [KF93]). Semantic token similarities are thereby derived from an English dictionary.

The plot of the cohesion over all document positions constitutes the *Lexical Cohesion Profile (LCP)*. Kozima presents the example plot in Figure 2.4. Minima of the smoothed plot are suggested to be chosen as text segments.

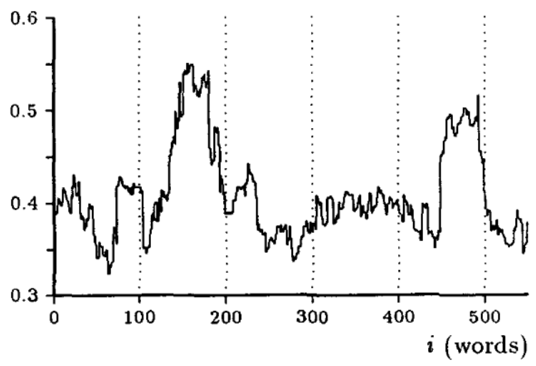


Figure 2.4: Lexical Cohesion Profile (adopted from Kozima)

Ponte and Croft: Local Context Analysis (1997)

Ponte and Croft [PC97] outline a dynamic programming solution for obtaining the optimum segmentation with respect to scores their algorithm assigns to all possible segments, i.e., to every sequence of 1 to N sentences within the document (where N is the maximum length of a segment).

Scores are calculated according to the results of a *Local Context Analysis (LCA)*, a query expansion method introduced by Xu and Croft [XC96] which is applied in order to generate an expanded set of features: Every sentence is used as a query for this method which then finds associated passages from a thesaurus. Words and phrases are extracted from the top 2000 passages and ranked according to their co-occurrence with the query terms. The top 100 features of this ranked list is then returned as result for the query.

A sequence of sentences is receiving a greater score

1. if intra-similarity of the LCA features of these sentences (which is calculated as the sum of the pairwise sentence similarities) is greater, or

2. if inter-similarity of the LCA features of all sentences in the sequence and the features of the neighbor sentences of the sequence (which is calculated as the sum of the pairwise sentence similarities of a fixed number of preceding – and following, respectively – sentences and the sentences of the sequence) is lower.

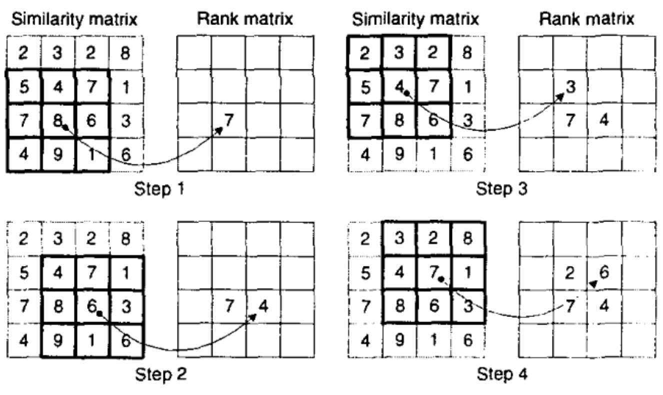
Similarity of two LCA feature sets is determined based on the number of features appearing in both sets. Using dynamic programming, the algorithm finally performs an optimal segment selection, maximizing the sum of segment scores.

2.3.4 Clusters/Partitions

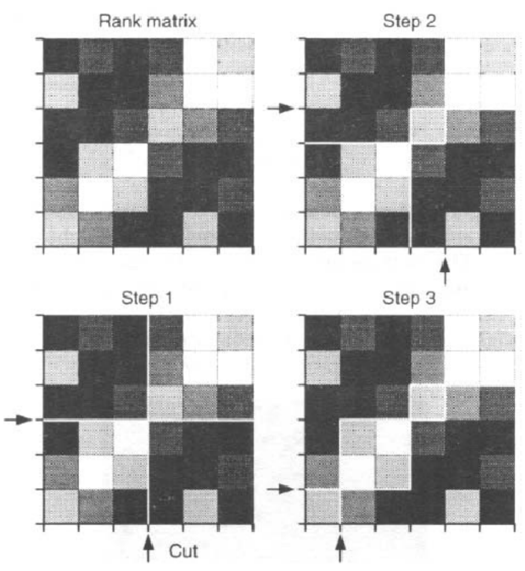
Choi: C99 (2000)

A frequently cited approach is Choi's *C99* [Cho00]. The algorithm first constructs a similarity matrix consisting of scores for each sentence pair where the similarity for a sentence pair is calculated using the cosine measure applied to the word stem frequency vectors of the sentences.

The resulting similarity matrix is then converted to a rank matrix where each entry is replaced by its rank in its local region (usually, an 11×11 sub matrix). Choi illustrates the creation of the rank matrix



(a) Creation of the rank matrix



(b) Divisive matrix clustering

Figure 2.5: Ranking and clustering (adopted from Choi)

with the example shown in Figure 2.5(a) where the size of the local comparison matrices is 3 × 3.

Segmentation is performed by applying a divisive clustering method to the rank matrix: Beginning with the segmentation which consists of only one cluster (the whole matrix, i.e. the whole document), the segmentation is refined iteratively by adding a boundary which maximizes the *inside density D* of the clustering which is defined as

$$D = \frac{\sum_{k=1}^{m} s_k}{\sum_{k=1}^{m} a_k}$$

where s_k is the sum of the matrix elements belonging to the k-th cluster, a_k is the number of matrix elements in the k-th cluster, and m is the number of clusters in the clustering.

All clusters are located along the diagonal of the matrix, thus, providing a simple mapping method from matrix clusters to document clusters: A cluster reaching from matrix entry (i,i) to (j,j) corresponds to a segment including sentences number i to j. With Figure 2.5(b), Choi presents an example of the working mechanism of clustering.

Malioutov and Barzilay: Minimum Cut Model (2006)

Malioutov and Barzilay present a graph-based approach [MB06]. The algorithm bases on the "max-intra-min-inter similarity" idea which is clarified by the authors with a sentence similarity plot of an example text (Figure 2.6): High density of blue dots indicates high pairwise sentence similarity (with respect to the cosine measure), red lines indicate true segment boundaries. The basis concept is to find the optimum "red lines", maximizing intra-similarity of segments, i.e., the density of blue dots within squares around the diagonal between the red lines (yellow), minimizing inter-similarity, i.e., the density of blue dots outside those square regions.

The algorithm of Malioutov and Barzilay implements this idea by finding the minimum cut of a graph consisting of one node for each sentence. Pairwise sentence cosine similarities are used as edge weights.

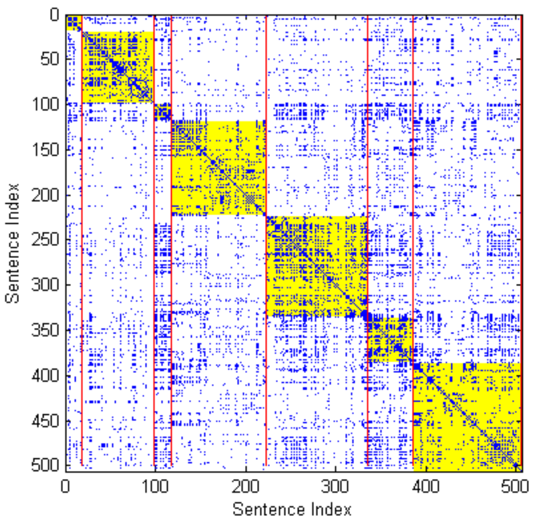


Figure 2.6: Sentence similarity plot for an example text (adopted from Malioutov and Barzilay and slightly adapted)
Blue dots indicate high similarity values of sentences, red lines are located at true segment boundaries. Yellow squares cover ranges which are in the same segment.

Thereby, the cosine measure is applied to smoothed word count vectors of the sentences. Each smoothed sentence vector is obtained by adding the vectors of adjoining sentences to it, weighted according to their distance.

After graph construction, the authors aim at finding the linear cut on that graph minimizing the *normalized cut criterion* for each pair of partitions (A, B):

$$Ncut\,(A,B) = \frac{cut\,(A,B)}{vol\,(A)} + \frac{cut\,(A,B)}{vol\,(B)}$$

$cut\,(A,B)$ denotes the sum of weights of edges between A and B, $vol\,(X)$ is the sum of weights of edges which have at least one node belonging to X. Thus, this criterion delivers lower values if the graph partition consists of subsets with high intra-similarity and low inter-similarity ($cut\,(A,B)$) between one subset and each other.[4]

A dynamic programming algorithm is applied for finding the optimal graph partition with respect to the normalized cut criterion. The found partition corresponds to a segmentation where sentences are in the same segment if they belong to the same subset of the partition.

2.3.5 Probabilistic Models

Utiyama and Isahara: U00 (2001)

Utiyama and Isahara suggest the probabilistic approach *U00* [UI01]. The authors define a probabilistic model and calculate the optimum segmentation according to this model. The concept bases on the

[4] Note the similarity to the approach of Ponte and Croft. However, the minimum cut model is in fact a global model whereas Ponte and Croft only consider inter-similarity between sentences and neighbor sentences, thus using a local model.

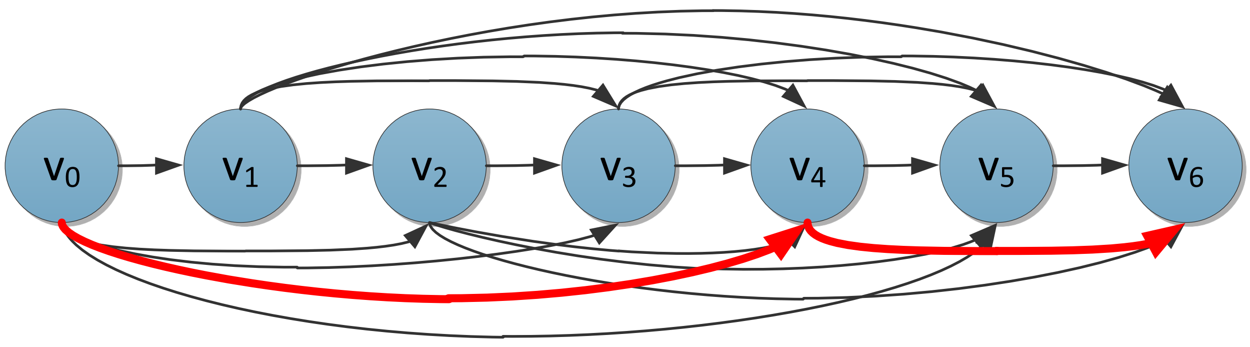


Figure 2.7: Construction of the graph (U00)

probability expression which is wished to be optimized: $P(S|W)$, where S is a certain segmentation and W is the text to be segmented. Application of Bayes' theorem yields the formulation

$$P(S|W) = \frac{P(W|S)P(S)}{P(W)}$$

For optimization of the left part of the equation, it is sufficient to optimize the numerator of the right part, since the denominator is constant for a given text W.

The model defines a priori estimates for the probabilities $P(W|S)$ and $P(S)$ where segmentations are assumed to be more likely if they contain less segments. For calculating probabilities, no knowledge is necessary as they are only based on word repetitions and not on semantic relationships.

The problem is then transformed to a minimum-cost path problem of a graph with nodes $v_0, v_1, ..., v_n$ (where n is the number of words in the text) and edges from each vertex to all vertices with a greater index. The cost of an edge (v_i, v_j) is set in a manner that greater costs indicate a lower probability of a segment consisting of words with indices $i+1, i+2, ..., j$. Therefore, the path from v_0 to v_n with minimal cost implicitly gives the solution for the most likely segmentation, according to this probability model. Figure 2.7 shows an example: If the selected (red and thick) edges make up the solution of the minimum-cost path problem, this corresponds to two segments, consisting of words 1 to 4 and 5 to 6.

Another probabilistic approach which claims to generalize the U00 model has recently been published by Eisenstein and Barzilay [EB08]. Therein, the authors impose a Bayesian model on the text segmentation problem and justify some of the methods used by Utiyama and Isahara, also generalizing them by introducing additional parameters. As an extension, also cue phrases are considered when searching for segment boundaries.

In a following work, Eisenstein extends this model for application on hierarchical text segmentation [Eis09].

2.3.6 Summary Table

Table 2.1 summarizes the most salient properties of the presented methods. The following properties are analyzed for each method:

- **Year**: The year of publication of the first paper the method has been presented in.
- **Authors**: The authors of this paper, or the first-named author if they are more than two.

- **Preprocessing**: List of utilities used as preparation of the actual algorithm. If parts of speech are listed in this column, these are the parts which are retained while others are rejected.
- **Thesaurus**: Contains "+" if the method relies on external knowledge pertaining to word meanings or relationships.
- **Token similarity**: The way on which similarity of tokens is calculated. "Similarity" may be a yes/no property (e.g., morphologically exact equality) or a numeric value indicating the degree of similarity (e.g., cosine similarity according to some token features). "–" means that the method does not compare tokens explicitly.
- **Sentence/block similarity**: The measure which is used for calculating similarity between sequences of tokens. "–" means that blocks are not compared explicitly.
- **Block cohesion**: The measure which is used for calculating the strength of cohesion of a sequence of tokens. "–" means that cohesion of blocks is not calculated.
- **Segment criterion**: The criterion which finally decides about the places of segment boundaries. Some algorithms use "depth scores" which are values assigned to each possible segment boundary (usually, each sentence boundary) according to the similarity of the adjacent blocks.
- **Algorithm**: Characteristic algorithms which are applied by the method.
  - *Fixed-sized window* denotes the approach of iterating over all token or sentence sequences of a certain length and, usually, calculating a score at each position.
  - *Lexical chains*: See Section 2.1.2.
  - *Dynamic programming* is a way to solve an optimization problem. Some segmentation methods apply it to generate an optimal segmentation with respect to a certain measure of "segment quality" which is often calculated for each possible segment.

2.4 Applications

As we have seen in the introduction, text segmentation is usually applied in order to ameliorate results of other algorithms which somehow rely on the topical structure of a text. In other words: text segmentation is, in most cases, not practiced for the sake of readability since longer texts of a certain quality are commonly segmented by their authors, e.g., using paragraphs or intermediate headlines. There are exceptions where texts are in fact not divided as it would be desirable for readability. This may be the case for the following types of texts:

- Transcripts of speech recordings
- Results of OCR[5] algorithms
- Articles of non-professional authors (e.g., in internet forums or Wiki-based platforms)

More commonly, however, text segmentation is applied before or in combination with other algorithms. The following list names some popular fields of application (+) and fields that one could think of to be appropriate for applying text segmentation (–):

- Text summarization (+)
- Information retrieval (+)

[5] Optical Character Recognition: Automatic transformation of images to text documents

Name	Year	References	Category	Preprocessing	Thesaurus	Token Similarity	Sentence/Block Similarity	Block Cohesion	Segment Criterion	Algorithm
Kozima: Lexical Cohesion Profile	**1993**	[Koz93]	Lexical Scores	n/a	+	Similarity w.r.t. dictionary descriptions	–	Specific dictionary exploitation method	Block cohesion	Fixed-sized window
Hearst: TextTiling	**1993**	[Hea93], [Hea94], [Hea97]	Lexical Scores	Stop word removal, n-grams	–	Morphological equality	Cosine similarity of token frequency vectors	–	Depth scores	Fixed-sized window
Okumura, Honda	**1994**	[OH94]	Lexical Chains	Nouns, verbs, adjectives	+	Equality of thesaurus category	–	–	Depth scores	Lexical Chains
Ponte, Croft: Local Context Analysis	**1997**	[PC97]	Lexical Scores	Usage of LCA features for each sentence instead of original words	+	–	Number of common LCA features for both blocks	Sum of pairwise sentence similarities	Block cohesion, inter-similarities of segments	Dynamic Programming
Choi: C99	**2000**	[Cho00]	Clusters/Partitions	Stop word and punctuation removal, Porter Stemmer	–	Morphological equality	Rank of cosine similarity of token frequency vectors	–	Clusters	Ranking, matrix clustering
Utiyama, Isahara: U00	**2001**	[UI01]	Probabilistic	Stop word and punctuation removal, Porter Stemmer	–	Morphological equality	–	–	Probability	Min-cost-flow graph problem
Galley et al.: LCseg	**2003**	[GMFLJ03]	Lexical Chains	Stop word removal, Porter Stemmer	–	Morphological equality	Cosine similarity of vectors with weights of overlapping lexical chains	–	Depth scores	Lexical Chains
Malioutov, Barzilay: Minimum Cut Model	**2006**	[MB06]	Clusters/Partitions	Stop word removal, Porter Stemmer	–	Morphological equality	Cosine similarity of smoothed token frequency vectors	–	Normalized cut criterion (high intra-similarity, low inter-similaity of segments)	Dynamic Programming
Ferret: F06T	**2007**	[Fer07]	Lexical Scores	Nouns, verbs, adjectives; lemmatization	–	Cosine similarity of co-occurent frequency vectors	Ratio of common tokens	Ratio of topically expressive tokens for the block	Block similarity, block cohesion	Fixed-sized window
Eisenstein, Barzilay	**2008**	[EB08]	Probabilistic	Stop word removal, Porter Stemmer	–	Morphological equality	–	–	Probability	Dynamic Programming
Marathe, Hirst	**2010**	[MH10]	Lexical Chains	Stop word and punctuation removal	+	Combination of co-occurrency and thesaurus categories [MH06]	–	–	Depth scores	Lexical Chains

Table 2.1: Overview of presented text segmentation methods

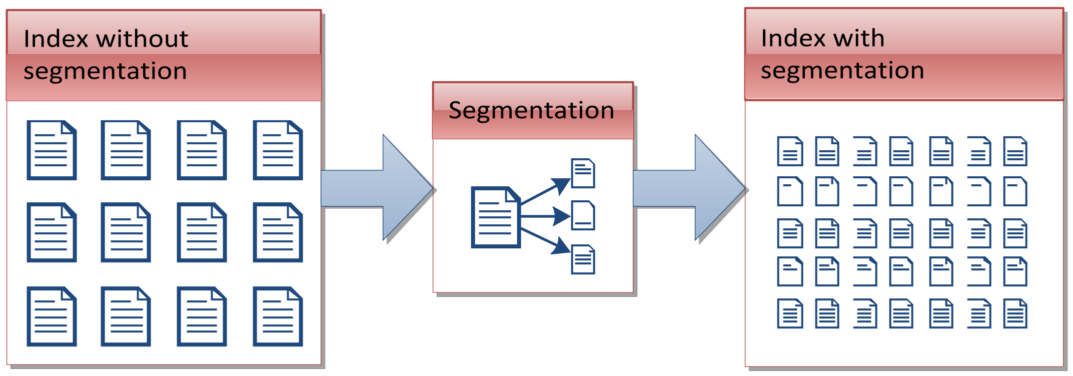


Figure 2.8: Text segmentation for enhancing information retrieval systems

- Language modeling (–)
- Hypertext linking (–)
- Anaphora resolution (–)

Each category is described below, enhanced with information on related work.

2.4.1 Text Summarization

Text summarization is the task of generating a rather short description of a text, providing a good overview of its contents. Text segmentation gives additional cues for the topic structure of a text and may therefore help in creating a summary covering all important topics of a text.

Barzilay and Elhadad [BE97] have included text segmentation into their summarization algorithm: They mainly derive important topics and significant sentences from lexical chains. However, they also apply a segmentation algorithm (namely, Hearst's TextTiling) and, during building the chains, they use the segments to separate the chains appropriately: A chain may only cross a segment boundary if it not only contains lexically related words on both sites, but also equal words. This leads to lexical chains which should better reflect the topical structure of the text, thus, optimizing the summary quality.

2.4.2 Information Retrieval

Information Retrieval denotes the task of finding documents which are relevant for a certain query. Text segmentation can support the user in not only finding the relevant document, but also the relevant segment of a relevant document. A possible strategy would be not to index whole documents but to segment them first and index the segments (see Figure 2.8). Alternatively, relevant documents might be segmented ad-hoc, presenting a relevant part to the user for each document.

Salton et al. [SAB93] suggest indexing documents not only at one single level of granularity, but on document, section, and paragraph level. For a query, similarities to all indexed elements are calculated, and most similar elements are presented to the user. This not only facilitates finding information for the user but also increases the recall: Many long documents contain only a small piece of relevant information, thus, exhibiting a low overall query similarity. Therefore, they are not returned for the query whereas, on section level, query similarity is much higher for the specific parts containing the relevant information which would likely appear in the result list. In some examples, the authors point up that the recall significantly improves if several levels of document granularity are combined for information retrieval.

Hearst and Plaunt [HP93] apply *TextTiling* (see Section 2.3.3) to segment texts according to their topical structure and index each document segment separately. In contrast to Salton, during retrieval, similarity values are only calculated for segments. However, the authors measure improvements of up to 28.2 % for both precision and recall.

2.4.3 Language Modeling

As mentioned by Reynar [Rey98], many NLP tasks are based on statistic language models. Speech recognition methods, for instance, often try to identify the most likely words according to the speech recording and the applied language model. Reynar suggests a language model which assumes words to occur with greater probability at a particular position if they are topically related to words of a certain environment of this position. To find topically related words for another word, this word is to be used as a query for an information retrieval system. The words within the results are assumed to be topically related. Now, if the results are rather segments than whole documents, the assumption of topical relatedness is more likely to be correct. Therefore, language modeling can be enhanced in this case by applying text segmentation to index building of the information retrieval system which is in fact exactly the approach we have mentioned in the previous section.

However, a system applying text segmentation methods to enhance language modeling is not yet available to the best of our knowledge.

2.4.4 Hypertext Linking

Hypertexts are texts that may contain links to other text or also inner links. A possible application of text segmentation would be to segment long texts and provide links between similar segments of the same document which would facilitate browsing through large documents since manually created documents often contain only few inner links (if any) since many authors only add links to other articles where appropriate, assuming that their own article is read linearly and, thus, abandoning inner links.

2.4.5 Anaphora Resolution

Anaphoras are words referring to words which have appeared previously in the text. Many NLP methods aim at resolving these references as this would contribute to the semantical expressiveness of the sentences containing anaphoras. According to Kozima [Koz93], a segmentation of the text can considerably facilitate anaphora resoltion: An anophara reference seldom crosses more than one segment boundary since any reader would be irritated by such long-range references. Therefore, the assumption that the referenced word is in the same segment or at least at the end of the previous one is reasonable and heavily reduces the set of possibly referenced words. To our knowledge, no study on the degree of improvement is available.

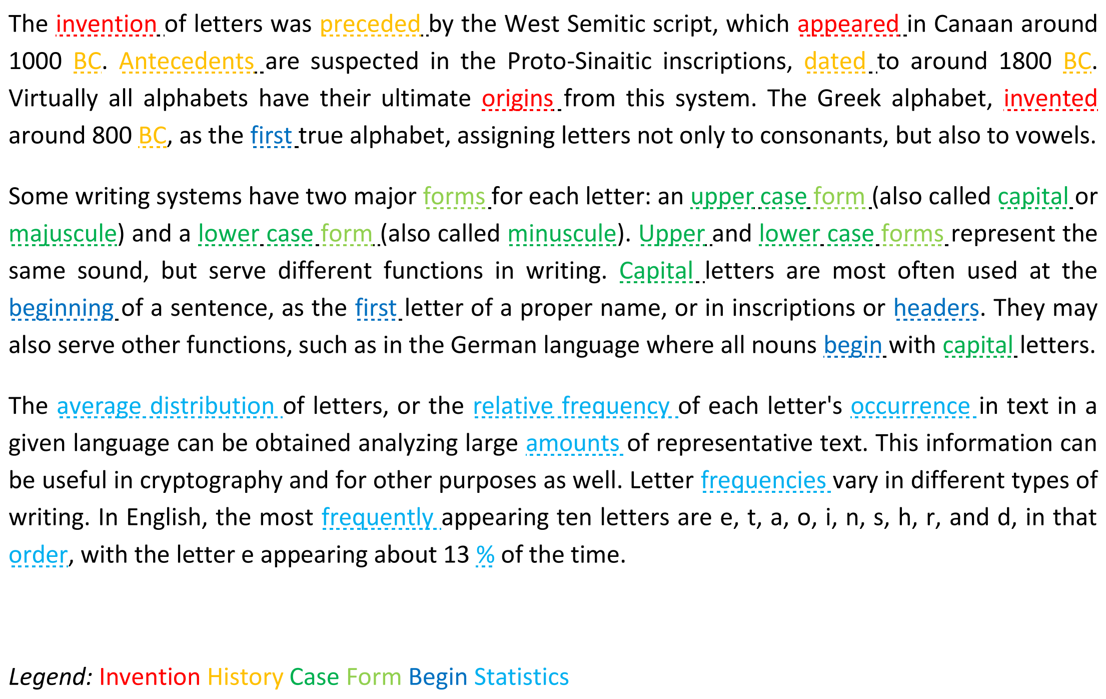
3 Graph-Based Algorithms for Text Segmentation

This chapter presents the approaches which have been developed in proceedings of the Bachelor Thesis. Their common property is that they consider texts as graphs in some stage of the algorithm: The first one considers the whole analyzed text as a graph of its terms which are related to each other through edges which indicate the semantic relatedness of the terms. The second one induces a subgraph on the text graph and aim at establishing segments which are represented as maximum compact semantic graphs. (For the definition of "compactness" in this respect, see Section 3.2.)

3.1 Cluster Blocks

The *Cluster Blocks (CB)* approach bases on our believe that a reasonable segmentation of a text in most cases coincides with clusters of semantically related words of the text.

To clarify this proposition, have a look at Figure 3.1. It shows a text on the topic "Letters" where tokens belonging to the categories "Invention", "History", "Case", "Form", "Begin", and "Statistics" are emphasized with different colors. We see clearly that distribution of words of that categories conforms well to the – reasonably chosen – paragraphs of the text: For instance, words belonging to the category "Invention", i.e., words which are semantically related with the term "invention" and the term "invention" itself, exclusively appear in the first paragraph whereas occurrence of "Statistics" terms is restricted to the last paragraph. If one would want to draw boundaries between the sentences, only according to categories of the emphasized terms, they would likely be the same as they are in the real division.


The invention of letters was preceded by the West Semitic script, which appeared in Canaan around 1000 BC. Antecedents are suspected in the Proto-Sinaitic inscriptions, dated to around 1800 BC. Virtually all alphabets have their ultimate origins from this system. The Greek alphabet, invented around 800 BC, as the first true alphabet, assigning letters not only to consonants, but also to vowels.

Some writing systems have two major forms for each letter: an upper case form (also called capital or majuscule) and a lower case form (also called minuscule). Upper and lower case forms represent the same sound, but serve different functions in writing. Capital letters are most often used at the beginning of a sentence, as the first letter of a proper name, or in inscriptions or headers. They may also serve other functions, such as in the German language where all nouns begin with capital letters.

The average distribution of letters, or the relative frequency of each letter's occurrence in text in a given language can be obtained analyzing large amounts of representative text. This information can be useful in cryptography and for other purposes as well. Letter frequencies vary in different types of writing. In English, the most frequently appearing ten letters are e, t, a, o, i, n, s, h, r, and d, in that order, with the letter e appearing about 13 % of the time.

Legend: Invention History Case Form Begin Statistics

Figure 3.1: Example text (adopted from the English Wikipedia article "Letter") clarifying the proposition that word clusters conform to segment boundaries

In other words: We can read the semantic structure of a text from the distribution of semantically coherent vocabular. Note that “semantical cohesion” also includes morphological equality as can be seen with the term “form” which has no directly related word in the text but constitutes an indicator of cohesion by itself. We call this indicator a *block* reaching from the first sentence containing a word of a semantic category to the last one. (We will define this in detail later.)

This idea of building blocks according to term clusters is very similar to lexical chains (see Section 2.1.2); however, construction of those blocks works different than construction of lexical chains: A lexical chain is usually iteratively extended by adding words which are lexically related to the original chain word whereas in our approach, we first consider the words of the text and do not build the blocks until we have collected the words to semantically related clusters which we assume to indicate text segments.

We believe that this order where term clustering is done in a first step may promote better results of segmentation. Each cluster is to indicate a significant topic of the text. Significant text topics and their vocabulary which is assumed to be collected in the clusters often have one of the following characteristics:

1. The topic is concentrated on a sequence of sentences constituting a rather short part of the whole document.
2. The topic’s vocabulary is spread over the whole document.
3. The topic is introduced at the beginning of the document and picked up in a longer passage later.

There may be other distributions of topic vocabulary; however, we believe that most distributions have one of these forms or at least a similar one. This allows to put together sentences of a topic to one (cases 1, 2) or two (case 3) or even more (a seldom case, though) blocks.

The blocks which are constructed this way should be a better indicator than lexical chains since a lexical chain always pertains to exactly one semantical concept whereas a block pertains to a topic which is assumed to be more appropriate to indicate segments: A single semantical concept may only span half a segment, topics tend to conform to segment boundaries as they may include several semantical concepts.

What is more, clustering of text terms is also text-sensitive, i.e., two words V, W which are in different clusters in text A may be in the same cluster in text B. This might happen, for instance, if text B contains some additional terms which are related both to V and W, leading to a cluster containing both V and W which indeed makes sense if many terms related to both words are existing. Lexical chains, on the contrary, use a fixed relatedness measure according to which words are added to an existing or a new chain and are therefore not sensitive to the actual content of a text that can cause two not obviously related words to belong to the same topic cluster.

3.1.1 Quick Overview

The algorithm has the following main steps after selection of relevant features:

1. Build the semantical graph (see Section 2.1.3) on the selected features.
2. Perform a clustering on the semantical graph, producing clusters with great pairwise semantical relatedness of features for every cluster.
3. For every (relevant) cluster, find sentences in the text containing terms of that cluster.

4. For every cluster, find blocks of sentences in the text containing terms of the cluster with high density.

5. Find sentence boundaries with high number of ending or beginning blocks and choose them as segment boundaries.

Figure 3.2 shows a schematic overview of the approach.

3.1.2 Feature Selection

Selection of expressive features is crucial for the approach. If many unexpressive, high-frequency terms are selected, they might, during clustering, quickly get arranged in lots of small unrelated clusters which can then lead to bad results of segmentation. We have decided to select nouns (proper as well as common nouns), verbs, and adjectives since we observed in concert with Halliday and Hasan [HH76] that other parts of speech such as pronouns, articles, or prepositions do not significantly contribute to the semantical contents of a text if the syntactical structure is ignored. In a further step, stop words such as auxiliary verbs are removed.

Besides selection of the features, we reduce them to common roots using lemmatization. This is indispensable for we would under no circumstances expect terms such as “knife” and “knives” to be assigned to different clusters which might happen in the case that the calculated semantical relatedness value for two different terms with equal lemma (e.g., “knife” and “knives”) is much lower than 1.

3.1.3 Building the Semantic Graph

In the second step, the document is considered as a graph where the previously selected features constitute the vertices and the semantic relationships between them constitute the edges. Note that we do not include double features in this virtual graph, i.e., if a term occurs twice or more frequently, it is anyhow represented only once in the graph. This is due to the idea not to weight different terms according to their frequency but to find topics of the text, whether they are more or less important.

We weight the edges with some kind of semantic relatedness as described in Section 2.1.3. Particularly, words are expected to be connected by an edge with maximum weight if they are semantically equal. One could also think of using other kinds of measures for the edges. For instance, simple co-occurrence-based values might be used, but we would expect significantly worse results for such a simple measure since it does not take long-range relations of words into account and therefore there would not be a good chance to find reasonable topical clusters in the graph.

Before the next step, a sparsification method is applied to the graph in order to remove edges with low semantic relatedness values. In our approach, we consider two variants of sparsification:

Threshold value: All edges with a value below a constant are removed.

Quota: A fixed quota of all edges is retained.

The sparsification is necessary because the clustering method applied subsequently works with unweighted graphs.

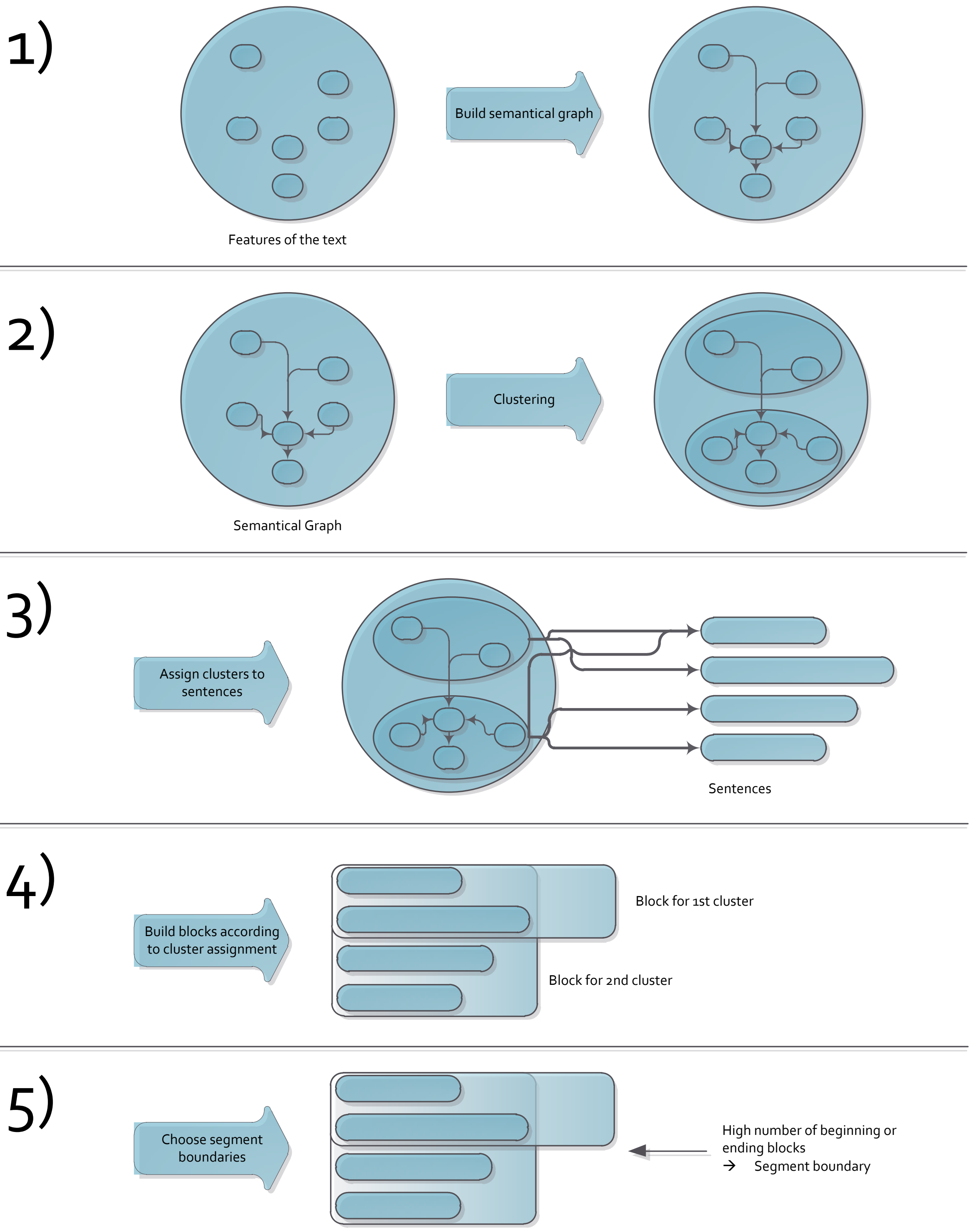


Figure 3.2: Cluster Blocks Algorithm

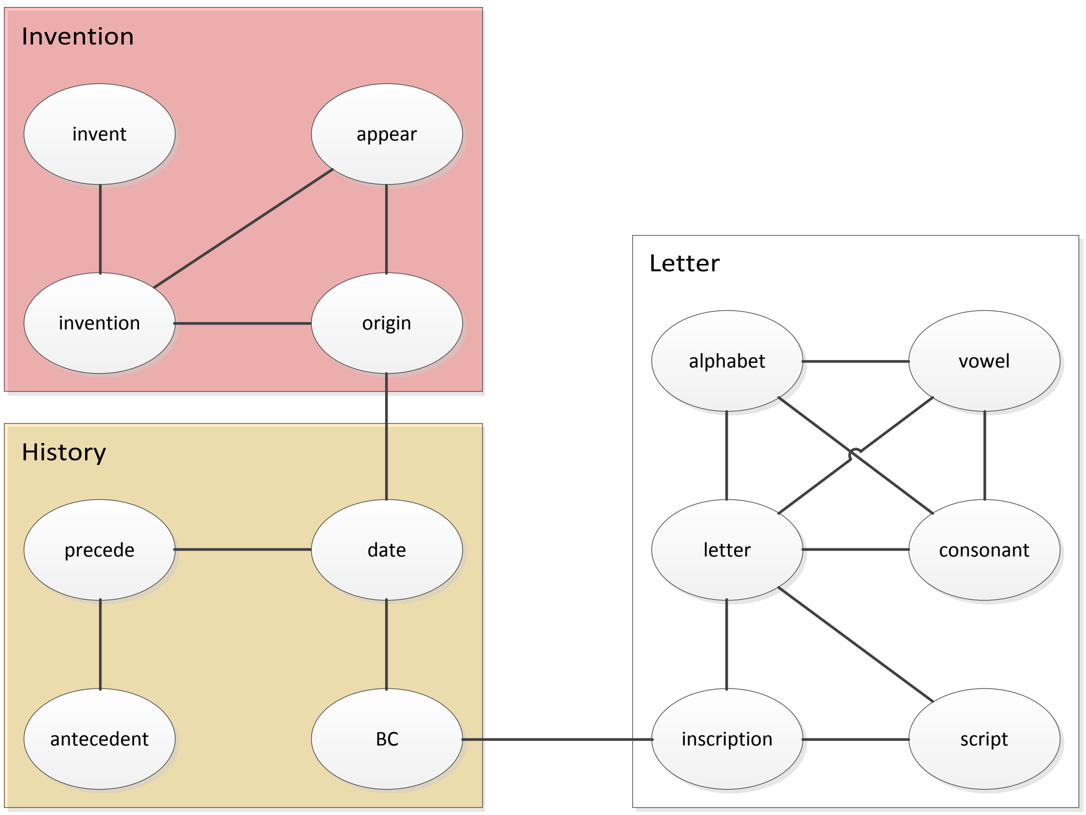


Figure 3.3: Example clustering for important terms in the first paragraph of the text shown in Figure 3.1

3.1.4 Graph Clustering

In the clustering step, we aim at finding clusters of topically related words. For topically related words should, among each other, have high semantic relatedness values, a clustering on the semantic graph is expected to serve the purpose. This rests upon a work of Grineva et al. [GGL09] who applied the same clustering algorithm as we will do to the semantic graph in order to find topical clusters of the text and then extract keyphrases from them.

Consider the first paragraph of the text from Figure 3.1. For this paragraph, we would expect a clustering similar to the one presented in Figure 3.3: Semantic relatedness values of edges between terms within one of the topics “Invention”, “History”, and “Letter” should be sufficiently high to make the edges endure the sparsification process so that the clustering will be able to detect the topical structure.

In short, we perform the clustering, according to Newman and Girvan [NG04], by removing edges with high *betweenness values* until all edges are removed. Whenever the connected components change during this process, we calculate a quality measure of the clustering, and finally we take the clustering with the most salient maximum within the quality measures.

In detail, the clustering algorithm consists of the following steps:

1. If there are no more edges in the graph, go to step 6. Else, go to the next step.
2. Calculate shortest paths (ignoring edge weights) between all vertices of the graph and count for each edge the number of shortest paths going along this edge. This value is called *betweenness*. If between a pair of vertices, not only one shortest path exists, but n paths, add $1/n$ to the counter of every edge for each path.

3. Remove the edge with the highest betweenness value.

4. Consider the connected components of the current graph as clusters and calculate the *modularity* of the clustering as suggested by Newman and Girvan: The modularity is the fraction of edges connecting vertices of the same cluster (in the original network where no edges have been removed) minus the expected fraction of such edges in a network with equal number of nodes and edges and same clustering where the edges are distributed randomly between vertices. It can be calculated using the formula $\sum_i \left(e_{ii} - a_i^2\right)$ where e_{ij} is the fraction of edges connecting vertices of clusters i and j and a_i is the fraction of edges having at least one endpoint in cluster i. This yields a value lower than 1 where 0 indicates a random clustering and values near to 1 indicate networks with strong community structure.

5. Go to step 1.

6. Smooth modularity values: For every three sequential values m_1, m_2, m_3, set $m'_2 := (m_1 + 2m_2 + m_3)/4$. Other smoothing masks are possible, but this one yielded the best results in our experiments in order to diminish insignificant local maxima while retaining significant ones.

7. Calculate *maximum strength* of modularity values. This is done by moving left from the initial value as long as values are decreasing, and calculating the difference of the lowest found value before stopping and the initial modularity value. The same is done on the right side and both differences are finally averaged to obtain the maximum strength. This procedure is in a way contrary to what Hearst does in her approach TextTiling to calculate depth scores [Hea94]. The rationale behind it is that the optimum clustering is often not the one with the absolutely maximum modularity value but the clustering which has a salient local maximum.

8. Choose the clustering with highest maximum strength.

The number of clusters with more than one term was originally planned to be chosen as number of segments because each topic is expected to have its own range in the text. Yet, this assumption could not be held because the number of clusters found did in experiments not correlate with the number of segments which might be due to the fact that in different documents topics are more or less woven and segments contain more or less different topics.

3.1.5 Finding Blocks

After having constructed the topical clusters, the algorithm tries to identify the positions in the text where the clusters are located. To this end, *blocks* of sequential sentences are identified for each cluster: Every sentence within such a block must contain at least one term of the cluster to belong to the block. There may be exceptions of sentences not containing a term if there are not more than a fixed number of such sentences in a row. This number is a parameter of the algorithm. Furthermore, blocks for which the number of sentences falls below a fixed threshold, another parameter, are discarded.

This idea of building blocks is in some respect similar to the construction of lexical chains as proposed by Morris and Hearst [MH91] (see Section 2.3.1): While Morris and Hearst use word categories from a thesaurus to represent the concepts of the chains, we use topics such as obtained from the clustering as representative of blocks. Topics are in our opinion more appropriate as concept representatives in text segmentation than thesaurus categories which are usually very fine-grained so that many lexical chains actually belong to the same topic. Thus, construction of lexical chains according to thesaurus categories might lead to deceptive chain boundaries as we have discussed in the introduction of Section 3.1.

3.1.6 Finding Segment Boundaries

In the last step, the algorithm uses the previously found blocks to decide where to place the segment boundaries. Similar to lexical chains again, sentence boundaries with a high number of beginning or ending blocks are considered to be segment boundary candidates.

Therefore, these *boundary strength* values are first counted and than smoothed: Whereas three sequential modularity values of clusters have been smoothed with the formula $m_2' := (m_1 + 2m_2 + m_3)/4$, in this case we weight the value of m_2 with 5 because tendency to insignificant local maxima is lower in this case as we observed in experiments. This is better respected using the smoothing mask $(1, 5, 1)$. The sentence boundaries with the highest smoothed boundary strength values are then chosen as segment boundaries. If the number of segments is not provided, the algorithm proceeds similarly to the TextTiling algorithm (see Section 2.3.3) and estimates the number according to the mean value μ and standard deviation σ of the boundary strength values: As experiments have revealed, in many cases, the number of boundaries which have a greater strength than $\mu + \sigma$ is a good guess for the number of actual segment boundaries.

3.1.7 Runtime Analysis

The following steps of the algorithm mainly contribute to its runtime complexity:

Clustering. Building the semantic graph in the JUNG framework (see Section 5.5) has complexity $O\left(T^2\right)$ where T is the number of relevant terms. Finding the optimum clustering with Girvan's algorithm is in $O\left(T^5\right)$: In each step which costs $O\left(T^3\right)$ one edge is removed until all edges have been removed, and the number of edges is in $O\left(T^2\right)$. Calculation of the modularity values for each possible clustering does not contribute significantly ($O(T)$). Picking out the best clustering by means of the modularity values is done by searching local maxima within those values. This additional effort has complexity $O(T)$ because T is a boundary for the number of clusterings.

Building blocks from the given clusters. This has complexity $O\left(T^2\right)$ because for each term every cluster is examined whether it matches the term, and the maximum number of clusters is T.

Calculating boundary strengths, i.e., the number of beginning and ending blocks for each sentence. For every cluster (up to T), there can be $\frac{S}{2}$ blocks where S is the number of sentences. This would be the case if every second sentence constitutes a new block for the cluster. There cannot be more blocks for one cluster because there must be a gap of at least one sentence between two blocks of the same cluster. Thus, runtime complexity of this step is $O(TS)$.

Finding segment boundaries by finding salient values of boundary strengths between sentences. For that purpose, the list of boundary strength values is sorted and the top values indicate boundaries. Thus, runtime complexity is $O\left(S \log S\right)$ for sorting.

Hence, the overall runtime complexity is $O\left(T^5 + T^2 + TS + S \log S\right) = O\left(T^5\right)$ because $O(S) = O(T)$ (see Section 3.2.6).

3.2 Maximum Compact Segments

The *Compact Segments (CS)* approach is built on the idea of compactness of coherent text ranges: Segments usually address a specific topic, thus containing vocabular which for the most part is highly cohesive in itself, i.e., most terms within a segment have a strong semantic relationship to some others within the same segment. Moreover, well chosen segments also have considerable semantical breaks to their neighbor segments because if they would not, one should rather unify them with the semantically similar neighbor sentences. This is why we try, in this algorithm, to maximize both compactness of segments and boundary strength.

To make this clearer, have a look at the example text in Figure 3.1 again: Terms like *form*, *case*, and *capital* in the middle paragraph are clearly semantically related. Contrariwise, they have only few commonalities with representative terms of the neighbor paragraphs such as *origin*, *distribution*, or *frequency*. Therefore, it seems to be a good idea to segment a text in a way which maximizes the inner semantic cohesion of paragraphs and minimizes the semantic relatedness to neighbor sentences. Admittedly, in the example text, there are also terms which seem to connect all paragraphs like *letter*, *script*, and *writing*. However, they make up only a small part of all terms of the text; hence, a segment comprising the whole text would not be very coherent.

Previous works have already tried to enhance text segmentation by maximizing internal cohesion and minimizing external similarity. For instance, Ponte and Croft use pairwise similarity of sentence features for calculation of both properties [PC97], and Malioutov and Barzilay merge both properties in one single value, the normalized cut criterion [MB06]. Our approach differs from recent works in that it does not consider sentences as units, but terms of the vocabulary in a set-of-words manner[1]. An advantage of this procedure should be that off-topic sentences within a segment do not harm its cohesion too much since the important factor of cohesion is, in our case, the whole vocabulary of a segment and not pairwise sentence similarities or the like. Therefore, a single off-topic sentence, adding not too much off-topic vocabulary, will not decrease cohesion significantly.

The advantage might be even greater for calculation of boundary strengths: Consider the case that a segment contains one sentence more at its end than it would be appropriate due to a topic shift. In a sentence-based comparison method, similarity of that segment and the following sentences is quite low because only one sentence, namely the last one, likely has a stronger relatedness to the following sentences. In our vocabulary-based approach, this one sentence might introduce lots of new terms which are related to the following sentences. Therefore, the similarity of the segment and the following sentences rises significantly if this one sentence is included. Consequently, we think that vocabulary-based comparison can lead to more accurate segment boundaries.

3.2.1 Quick Overview

A schematic overview is presented in Figure 3.4.
The algorithm has the following main steps:

1. Divide the text into blocks. Each block consists of (nearly) the same number of sentences.
2. For each segment candidate (any sequence of blocks), calculate its segment quality:
   a) Induce the semantic graph for the vocabulary of the candidate (step 1 in Figure 3.4).
   b) Calculate the compactness of the graph as defined in Section 3.2.4 (step 2a in Figure 3.4).
   c) Calculate the boundary strength of the candidate as defined in Section 3.2.4 (step 2b in Figure 3.4).
   d) Merge compactness and boundary strength values to a segment quality value.

[1] A term is in the “set of words” if it appears in the text, regardless of the number of its occurrences.

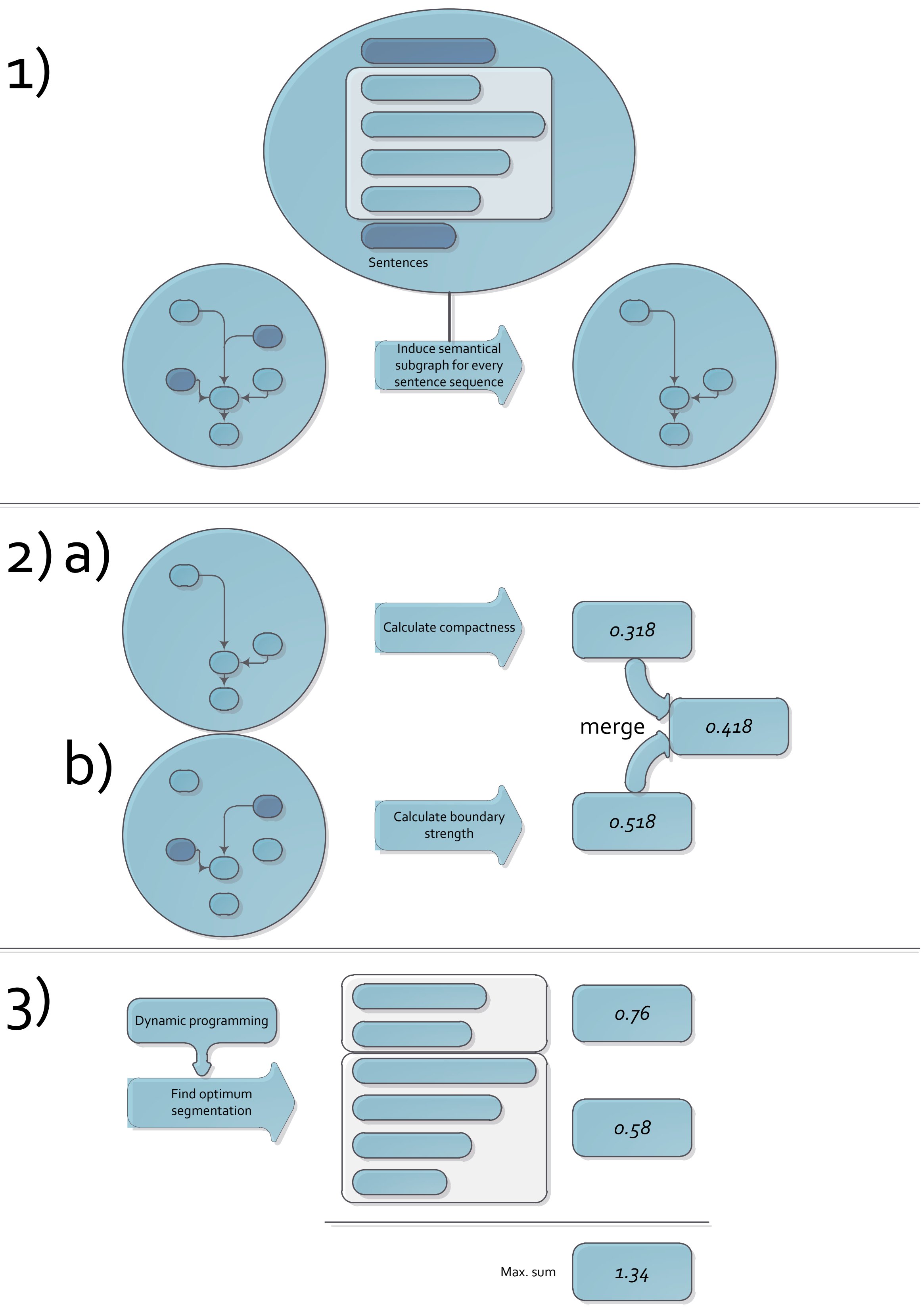

1)
Sentences
Induce semantical subgraph for every sentence sequence
2) a)
b)
Calculate compactness
Calculate boundary strength
0.318
merge
0.418
0.518
3)
Dynamic programming
Find optimum segmentation
0.76
0.58
Max. sum
1.34

Figure 3.4: Compact Segments Algorithm

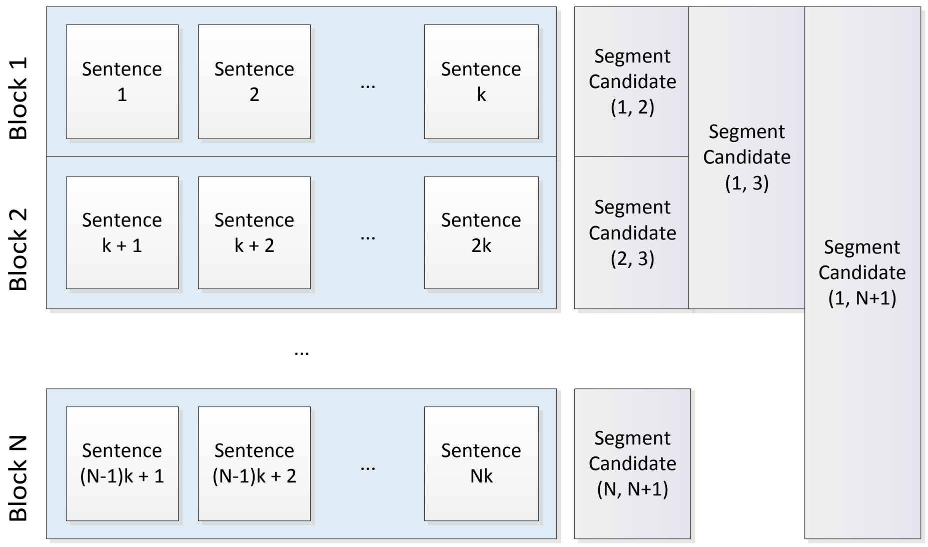


Figure 3.5: Partition of the document into blocks and possible segment candidates

3. Find the optimum segmentation of the text which maximizes the sum of quality values of the chosen segments.

3.2.2 Division into Blocks

Conceptually, each block should always have a length of one sentence. However, runtime must be taken into consideration: As this approach is going to compute quality values for each possible segmentation (where blocks are the smallest unit), runtime and memory usage is proportional to $\frac{N(N+1)}{2}$ where N is the number of blocks. Due to this quadratic runtime behavior, number of blocks is limited to 100. Hence, this algorithm will never produce more than 100 segments.

If the number of sentences S is greater than 100, each block is guaranteed to contain exactly $\frac{S}{100}$ sentences if S is a multiple of 100. If it is not, each block contains either $\left\lfloor \frac{S}{100} \right\rfloor$ or $\left\lceil \frac{S}{100} \right\rceil$ sentences.

3.2.3 Segment Candidates

The approach assigns a quality value to every possible segment which is called a *segment candidate*. A segment candidate is any sequence of blocks in the text, e.g., 1st block to 2nd block, 1st block to 3rd block, and 2nd block to 3rd block. Figure 3.5 illustrates the partition of the document into N blocks, each with k sentences. It furthermore shows ranges of possible segment candidates: E.g., the candidate $(1,3)$ covers the first and second block.

3.2.4 Segment Quality

For each segment candidate, a quality value Q is calculated indicating whether the candidate should indeed constitute a segment. Positive values represent a candidate which is appropriate for being taken as a segment, negative values suggest not to make the candidate a segment.

The quality value is calculated by the use of two criteria:

Compactness: The compactness of a candidate is a measure for the inner semantic similarity. Thus, a candidate containing information about different topics has a lower compactness than a candidate which is only about a single topic.

Boundary Strength: The boundary strength quantifies the semantic similarity of the candidate and its adjacent sentences. The higher this value is, the less similar are the vocabulary of the candidate and the terms of the adjacent sentences.

Both criteria are wished to be maximized since we aim at segments which talk about a single topic and which significantly differ from adjacent sentences. At the same time, one of these criteria alone would very likely not be sufficient: If we would only concentrate on the compactness, this would possibly yield too many very short segments which are – just due to their shortness – very compact, though not having notable semantic boundaries because adjacent sentences may still concern the same topic. If we would, contrarily, only consider the boundary strength, this would possibly yield too many very long segments which expose strong semantic boundaries, but contain different topics. This is why we propose a combination of both criteria.

For combination of both criteria, we define the quality function q which calculates the quality value Q for given compactness and boundary strength values C and B and the number of sentences L the candidate comprises:

$$Q = q(C, B, L)$$

We will analyze multiple variants of q which all have the following form:

$$q(C,B,L) = \alpha L^{\beta} C + (1-\alpha) L^{\gamma} B - d \tag{3.1}$$

α is a weighting factor between 0 and 1 which constitutes the weight with which the compactness influences the quality. Correspondingly, $1-\alpha$ indicates the weight of the boundary strength. β and γ can be seen as length recompense factors for compactness and boundary strength part, respectively: If two segment candidates have the same compactness value, but the second one has double length, it should likely receive a greater score. Thus, β and γ will most likely have a value greater than 0 in the optimum configuration.

d is an additional penalizer. For a constant number S of segments, the total penalty has a constant value of Sd; hence, d does not impact the optimum segmentation for a given number of segments. Instead, its function is to tune the number of segments: The higher its value, the less segments will be in the optimum segmentation and vice versa.

The idea of rating possible segments is not new: Malioutov and Barzilay [MB06] have defined their Minimum Cut Criterion which rates possible partitions of the text according to which the final segmentation is selected (see Section 2.3.4). They also consider a kind of boundary strength which is the "cut" in their model, but do not calculate a value for compactness which, in our opinion, should enhance results because too big segment candidates with strong boundaries, but different covered topics receive too good scores if the concept of compactness is ignored. The next two subsections define in detail the ingredients of our segment quality measure.

Compactness

The compactness as the first criterion of segment quality of a segment candidate is defined as the compactness of the semantic graph induced by the features[2] of the candidate. It is to express the strength

[2] As for Cluster Blocks, we suggest lemmas of nouns, verbs, and adjectives to be used as features. Stop words are removed, as well.

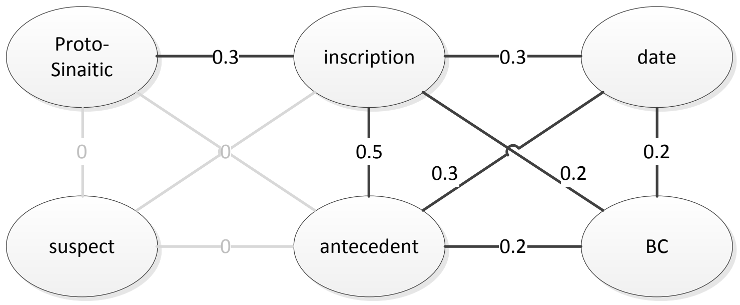


Figure 3.6: Semantic graph of a segment candidate

of semantic cohesion of the candidate. E.g., if all edges within the semantic graph are 1, the semantic cohesion will have maximum possible value, whereas many edges close to 0 will yield a low compactness value. We now describe the detailled calculation of compactness:

In a first step, the semantic graph for the segment candidate is built. (Figure 3.6 gives an example. Note that some 0-value edges are left out in the figure since the semantic graph is actually complete.) Compactness is then calculated according to a work of Egghe and Rousseau from 2003 [ER03] which extends the compactness measure of Botafogo et al. [BRS92] for application to weighted graphs. Therein, compactness of an undirected weighted graph, such as a semantic graph, is defined as

$$C = \frac{MAX - \sum_{1 \le i < j \le V} d\left(i, j\right)}{MAX - MIN} \tag{3.2}$$

where $d : \{1, 2, \ldots, V\}^2 \to [a, b]$ is a distance function indicating the dissimilarity of vertices, MAX is the maximum possible sum of all edge weights, MIN is the minimum possible sum[3], and V is the number of vertices. Values of C close to 1 indicate a very compact graph, thus, having many edges with a low distance value close to a, and values close to 0 indicate a very incompact graph.

Since in a semantic graph weights w_{ij} indicate similarity rather than dissimilarity, we define:

$$d(i, j) := 1 - w_{ij}$$

The maximum sum of edge weights in the graph is achieved if $d(i, j) = 1$ for all edges (i, j). The minimum sum occurs if $d(i, j) = 0$ for all edges (in case of $w_{ij} = 1$). As the number of edges in a complete graph with V edges is $\frac{V(V-1)}{2}$, we set:

$$MAX := \frac{V(V-1)}{2}$$
$$MIN := 0$$

Applying this to equation (3.2) yields:

$$C = \frac{2}{V\left(V-1\right)} \sum_{1 \le i < j \le V} w_{ij}$$

This is exactly the average edge weight of the semantic graph.

[3] The maximum possible sum happens if all edges have value b. Analogously, the minimum sum occurs for distance a on all edges.

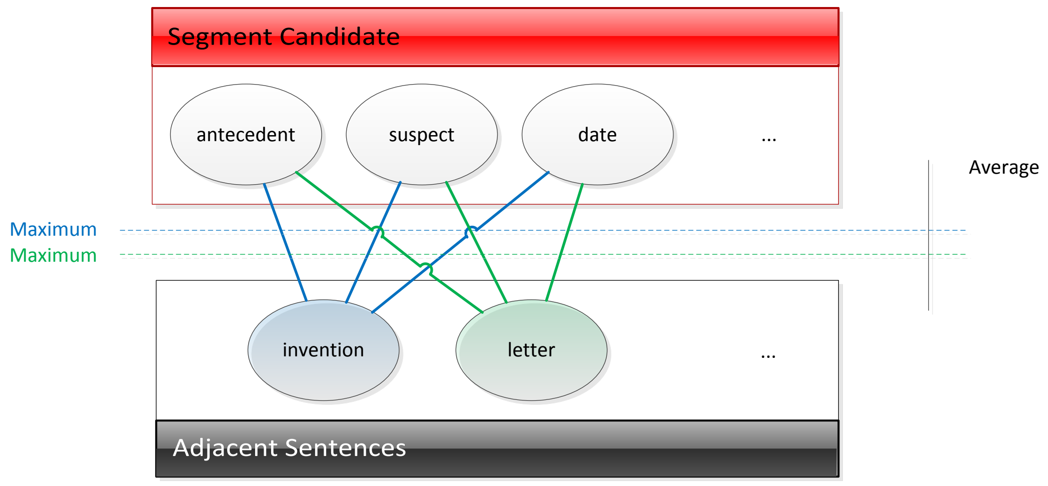


Figure 3.7: Calculation of the boundary strength

Example. Consider the graph in Figure 3.6 to represent the terms which are present in a certain segment candidate. In order to calculate compactness of this candidate, the average edge weight of this semantic subgraph has to be evaluated. Since the graph contains 6 vertices and is actually complete, it has $\frac{6 \cdot 5}{2} = 15$ edges. Hence, compactness is:

$$C = \frac{0.5 + 0.3 + 0.3 + 0.3 + 0.2 + 0.2 + 0.2}{15} = \frac{2}{15} = 0.1\overline{3}$$

Boundary Strength

The second criterion of segment quality of a segment candidate is the boundary strength B. It is to indicate the dissimilarity of a segment candidate and its adjacent sentences which are defined as the S_{adj} preceding and S_{adj} subsequent sentences of the segment candidate. If there are not that many preceding or subsequent sentences, the set of adjacent sentences will be smaller, accordingly.

The boundary strength is defined as:

$$B = 1 - \frac{\sum_{i=1}^{T_{adj}} s_i}{T_{adj}}$$

B is to be understood as complement of the average semantic relatedness of the terms of the adjacent sentences to the segment candidate. T_{adj} is the number of terms in the adjacent sentences. s_i is the semantic relatedness of the i-th term of the adjacent sentences to the segment candidate. It is calculated as the maximum relatedness of the term to any term of the segment candidate because this turned out to be a significant indicator of the actual topical relatedness whereas the average of all relatedness values to the terms of the segment candidate was not very expressive in our experiments.[4] If the i-th term is contained both in the adjacent sentences and in the segment candidate, its relatedness value s_i is set to 1.

Figure 3.7 illustrates calculation of boundary strengths: The red box contains the terms of the currently analyzed segment candidate, the black box contains the terms of its adjacent sentences. Dotted lines indicate maxima of weights of edges with the same color. Note that the average value which is calculated as shown in the figure is not the final boundary strength value; this is obtained by subtracting

[4] This is more like Malioutov's and Barzilay's MinCut model defines the boundary strength which takes into account all edges between the segment candidate and the rest of the document.

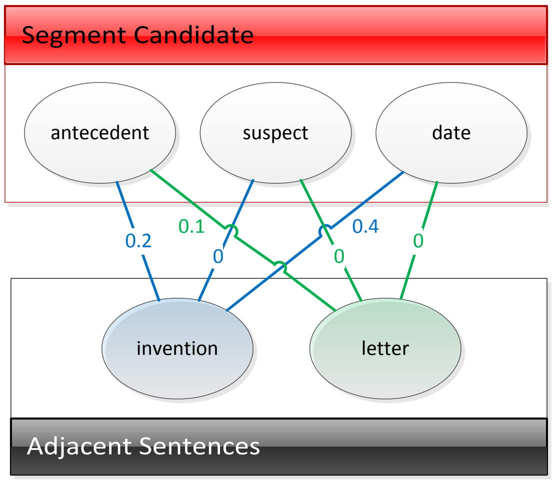


Figure 3.8: Example for calculation of the boundary strength

that average value from 1.

Example. Now consider the semantic relatedness values given in Figure 3.8 and assume that there are not more terms in the currently analyzed segment candidate and its adjacent sentences, respectively, than shown in the figure. (This is not a realistic assumption. However, for this example, it serves the purpose.) In order to calculate the boundary strength of this segment candidate, one has to find the maximum relatedness value for each term of the adjacent sentences. In this case, we identify the values 0.4 for "invention" and 0.1 for "letter". These values must then be averaged (0.25). The difference of 1 and this average is the boundary strength:

$$B = 1 - 0.25 = 0.75$$

3.2.5 Dynamic Programming for Optimization

The final step of the algorithm is finding the optimum segmentation with the maximum sum of segment quality values. For that purpose, dynamic programming is applied as already done in similar ways by Ponte and Croft [PC97] and Malioutov and Barzilay [MB06]. The reader may skip this subsection if she is not interested in the details of dynamic programming.

Formally, we define the optimization problem as follows: For given segment candidate quality values $Q_{ij} \in \mathbb{R}$ denoting the segment quality of blocks i to j (excluding) and the number of blocks N, find the segmentation Seg which maximizes the target function t:

$$t\left(Seg\right) = \sum_{i=0}^{\left|Seg\right|} Q_{Seg(i)Seg(i+1)}$$

The solution space Ω is defined as $\Omega = \mathscr{P}\left(\{2, 3, \ldots, N\}\right)$ where $\mathscr{P}$ denotes the power set of its argument. A given segmentation $Seg \in \Omega$ is meant to contain the indexes of blocks which begin a new segment, except for the first block which always begins a segment. E.g., for $N = 10$, $Seg = \{4\}$ describes the segmentation with two segments consisting of 1st to 3rd and 4th to 10th block.

$Seg\left(i\right)$ denotes the i-th greatest element in Seg if $1 \leq i \leq \left|Seg\right|$, and additionally we define:

$$Seg\left(0\right) := 1$$
$$Seg\left(\left|Seg\right| + 1\right) := N + 1$$

Optimization is performed with dynamic programming. For that purpose, we define two $N \times N$-matrices V and W where v_{ij} will contain the segment rating for an optimum segmentation of the first i sentences into j segments and w_{ij} will carry the number of sentences which are contained by the first $j-1$ segments of that optimum segmentation. Hence, W will allow the construction of the optimum segmentation after the solution is found.

The matrices are initialized as following:

$$\begin{aligned} \forall i: \quad v_{i1} &:= Q_{1(i+1)} \\ \forall i \; \forall j \neq 1: \quad v_{ij} &:= null \\ \forall i \; \forall j: \quad w_{ij} &:= null \end{aligned}$$

Afterwards, optimum segmentations are calculated for every possible number of segments J. Thus, for $J := 2, 3, \ldots, N$, it is set:

$$\begin{aligned} \forall i \geq J: \quad v_{iJ} &:= \max_{J-1 \leq k < i} \left(v_{k(J-1)} + Q_{(k+1)(i+1)} \right) \\ w_{iJ} &:= \operatorname{argmax}_{J-1 \leq k < i} \left(v_{k(J-1)} + Q_{(k+1)(i+1)} \right) \end{aligned} \tag{3.3}$$

The optimum segmentation Seg_{nc} for the first n blocks and a given number of segments c can than be constructed by using the back-pointers in W:

$$Seg_{nc} = \begin{cases} \emptyset & c = 1 \\ \{w_{nc} + 1\} \cup Seg_{w_{nc}(c-1)} & c > 1 \end{cases} \tag{3.4}$$

If the optimum number of segments c^* is to be obtained using the quality values, it can be calculated as:

$$c^* = \operatorname{argmax}_{1 \leq c \leq N} v_{Nc} \tag{3.5}$$

With (3.4) and (3.5), we can get the overall optimum segmentation Seg^* with the following formulation:

$$Seg^* = Seg_{Nc^*}$$

3.2.6 Runtime Analysis

The runtime of the algorithm is mainly influenced by the following steps:

Calculation of compactness values for each possible segment. This has a runtime complexity of $O\left(NT^2 + N^2\right)$ where N is the number of blocks and T is the number of terms in the document: For every group of segment candidates starting with a certain block, all terms of subsequent blocks are successively added to the semantic graph (this is the term NT^2); the number of segment candidates, however, is only bounded by N^2.

Calculation of boundary strength values for each possible segment. The complexity is $O\left(N^2 T\right)$ because terms of the currently analyzed segment candidate and adjacent terms have to be collected for each candidate. The number of adjacent terms may be considered to be constant since the number of adjacent sentences is constant; however, the number of terms within the current candidate is only bounded by T. The number of possible candidates is $\frac{N(N+1)}{2} \in O\left(N^2\right)$.

Application of function q for merging compactness and boundary strength values. This has constant runtime for every segment candidate and, thus, is in $O\left(N^2\right)$.

Finding the optimum segmentation with dynamic programming. As we have separated this component, it uses sentences as smallest possible unit. As can be seen in equation (3.3), runtime complexity is therefore in $O\left(S^3\right)$ where S is the number of sentences.

This yields a total runtime of $O\left(NT^2+N^2+N^2T+N^2+S^3\right)=O\left(NT^2+N^2T+S^3\right)$. As we can say that, in average, a sentence contains a certain number of terms, it holds:

$$T=cS \Rightarrow O(T)=O(S)$$

Therefore, the total runtime complexity can be formulated as $O\left(NT^2+N^2T+T^3\right)$. For a given maximum number of blocks we have $O\left(T^3\right)$. If the maximum number of blocks is not provided, N is bounded by S. Hence, the runtime complexity is not different in this case.

4 Evaluation

Text segmentation algorithms have a weakly defined goal: dividing texts into “topically coherent” segments. This is what makes comparison of different algorithms difficult: Which of two segmentations can be said to be “better” for a certain text? How can quality of segmentations be measured objectively? And what kinds of texts can be used to test segmentation algorithms?

This chapter tries to answer these questions and presents the measures we have used to evaluate our algorithms in a comparable way. It furthermore presents a new corpus for evaluation and, finally, the results of our evaluations.

4.1 Methodology

The main idea of most frequently applied evaluation methods in text segmentation is based on comparison of segmentations as calculated by the algorithms to be evaluated and “gold standard” segmentations. A gold standard segmentation for a certain text defines places within this text where actual topical boundaries occur. The more (nearly) correct boundaries an algorithm detects, the better it is rated in evaluation. However, definition of a measure which fairly quantifies this connection is not trivial. We will use P_k and WindowDiff metrics to evaluate our algorithms on different corpora and, due to the popularity of those metrics, will be able to compare results to those of existing algorithms. Refer to Section 4.2 for the metrics we use and to Section 4.3 for the corpora. The rest of this section pertains to formal definition of our evaluation process.

Formally, we define an evaluation as following: An evaluation E of a given text segmentation algorithm A on a given corpus C is defined as tuple $E = (A, C)$. The corpus itself consists of a set of texts T_C and a function seg_C which assigns a segmentation to each text in T: $C = \left(T_C, seg_C\right)$. Hence, for $t \in T$, $seg_C(t)$ is the gold standard segmentation for text t in corpus C. A segmentation Seg is formally defined as $Seg \subseteq \mathbb{N} \setminus \{0,1\}$ with

$$i \in Seg \Leftrightarrow \text{The } i\text{-th unit begins a new segment and } i > 1.$$

The considered unit depends on the context; it may be tokens or sentences, for instance. The first unit has index 1 and is never element of Seg since the first segment always begins with the first unit, regardless of the type of units. When we write $Seg(i)$, we refer to the i-th greatest element of Seg. Additionally, we define $Seg(0) := 1$, and $Seg\left(\left|Seg\right| + 1\right) := \infty$.

The function $A(t)$ denotes application of algorithm A to text t and yields a segmentation of the text. Accordingly, the function $A(T)$ denotes application of the algorithm to a set of texts T and yields a function which assigns the calculated segmentation to each text in T.

For a given quality metric m, we define the equally named function $m\left(t, Seg_1, Seg_2\right)$ which yields the result of the metric for a given text t, the gold standard segmentation Seg_1 and a calculated segmentation Seg_2. The function $m_\oslash(E)$ denotes the average result of the metric for the given evaluation definition E:

$$m_\oslash(E) = m_\oslash(A, C) := \frac{\sum_{t \in T_C} m\left(t, seg_C(t), A(t)\right)}{\left|T_C\right|}$$

4.2 Metrics

The goal of evaluation metrics in text segmentation is to obtain a comparable quality value for the segmentation of a certain text with respect to a gold standard segmentation. Therefore, correct segment boundaries should be rewarded, wrong or missing boundaries should be penalized. Early works on text segmentation methods, e.g., Hearst's presentation of TextTiling [Hea94], used *precision* and *recall* measures which are famous to rate information retrieval systems. Applied to text segmentation, precision would denote the quota of gold standard boundaries within the set of calculated boundaries, and recall denotes the quota of found gold standard boundaries with respect to all gold standard boundaries.

However, as pointed out by Beeferman et al. [BBL97], recall and precision are not quite appropriate for evaluation of text segmentation algorithms which we will explain with the following example: Consider the text presented in Figure 3.1. It consists of three paragraphs, each containing four sentences. Thus, the gold standard segmentation with sentences as units would be $Seg = \{5, 9\}$. If we further proceed on the assumption that a segmentation algorithm yields the segmentation $Seg = \{4, 8\}$, i.e., the last sentences of the first and second paragraphs have wrongly been assigned to the second and third segment, respectively, this would lead to precision and recall values 0. This is not what is desired in this case since the algorithm has at least detected the correct number of segments and it missed correct boundaries only closely. Hence, it should be rated better than an algorithm which, for example, does not detect any boundary at all. This is the point where recall and precision fail.

4.2.1 P$_k$

As a new metric which should overcome the disadvantages of recall and precision, Beeferman et al. proposed P$_k$ [BBL97], [BBL99] which also takes into account near misses: The metric moves a sliding window over the text – either on token or on sentence base – which has a first pointer and a second pointer which point at units with a distance of k. E.g., if $k = 2$ and sentences are considered as units, the pointers will iterate over the sentence pairs $(1,3),(2,4)$, etc. For each analyzed pair of units (i, j), three cases are distinguished:

1. Units i and j are in the same segment, both in the calculated and the gold standard segmentation.
2. Units i and j are in the same segment in one segmentation, but in different segments in the other one.
3. Units i and j are in different segments in both calculated and gold standard segmentation.

Case 2 is considered as an *error*, other cases are not because in case 2, calculated and gold standard segmentations differ in that the one puts a boundary between the two units whereas the other does not. The number of errors is then counted while the window moves over the text, and the result is defined as the quota of errors with respect to the number of possible errors:

$$P_k\left(t, Seg_1, Seg_2\right) := \frac{\sum_{i=1}^{|t|-k} error\left(i, i+k, Seg_1, Seg_2\right)}{|t| - k}$$

$$error\left(i, j, Seg_1, Seg_2\right) := \begin{cases} 0 & \text{if } sm\left(i, Seg_1\right) = sm\left(j, Seg_1\right) \wedge sm\left(i, Seg_2\right) = sm\left(j, Seg_2\right) \\ 0 & \text{if } sm\left(i, Seg_1\right) \neq sm\left(j, Seg_1\right) \wedge sm\left(i, Seg_2\right) \neq sm\left(j, Seg_2\right) \\ 1 & \text{if } sm\left(i, Seg_1\right) = sm\left(j, Seg_1\right) \wedge sm\left(i, Seg_2\right) \neq sm\left(j, Seg_2\right) \\ 1 & \text{if } sm\left(i, Seg_1\right) \neq sm\left(j, Seg_1\right) \wedge sm\left(i, Seg_2\right) = sm\left(j, Seg_2\right) \end{cases}$$

$$sm\left(k, Seg\right) := \left|\left\{b \in Seg | b \leq k\right\}\right| + 1$$

The $error$ function returns 1 in case 2 we discussed above, and the sm function returns the index of the segment the given unit belongs to.

The P_k measure can also be understood in a probabilistic manner as it indicates the probability that two units of the text with a distance of k are inconsistently classified. Thus, P_k heavily depends on the value of k: Values are becoming much better for small values of k because the probability of inconsistent classification of two units in two segmentations is much lower if their distance is smaller. For comparability, k must be standardized in some way. For that purpose, Beeferman suggests k to be set to half the average segment size according to the gold standard segmentation. E.g., if a text is analyzed on token-basis and contains 120 tokens and 4 segments, for evaluation with P_k, k must be set to $\frac{1}{2} \cdot \frac{120}{4} = 15$.

An example for the calculation of the P_k value is presented in Figure 4.1: In the first step, the first and the third unit are compared with respect to their segment membership. As they belong to the same segment in both segmentations, no error is counted in this step. Contrariwise, the second step reveals an error as defined in the previous case distinction because in the hypothesized segmentation, the second and fourth units belong to the same segment while in the gold standard, there is a segment boundary before the fourth unit. In the third and last step, there is no error again because units are in different segments on both sides. The result for this example would be the number of errors divided by the possible number of errors: $\frac{1}{3} = 33.\overline{3}\%$.

4.2.2 WindowDiff

As an improvement of P_k, Pevzner and Hearst suggest the WindowDiff metric [PH02]. They argue that P_k penalizes false negatives more heavily than false positives and that it overpenalizes near misses. Moreover, the number of segment boundaries is ignored in some cases. We will have a look at these disadvantages by analyzing them for a simple example taken from the article “Vega” of the English Wikipedia:

> *Vega is the brightest star in the constellation Lyra, the fifth brightest star in the night sky and the second brightest star in the northern celestial hemisphere. Only Arcturus is brighter.*
>
> *Astronomers term “metals” those elements with higher atomic numbers than helium. The metallicity of Vega’s photosphere is only about 32% of the abundance of heavy elements in the Sun’s atmosphere.*

This text contains 4 sentences, 62 tokens, and a gold standard segment boundary at the paragraph. If we analyze this text on token base, the k parameter of P_k would be set to 15 because 31 is the average segment length. Below, we suggest some examples where the disadvantages of P_k become obvious:

Lax penalization of false positives: If a segmentation algorithm does not find any segment boundary, 15 errors would occur during the P_k sliding window method; thus, $P_k = \frac{15}{47} \approx 31.9\%$. If, however, a segmentation algorithm would find a boundary after the first and the second sentence instead (false positive after the first sentence), only 4 errors are found, namely when the second pointer of the window is over the second sentence. This yields a total P_k value of $\frac{4}{47} \approx 8.5\%$. Thus, false positives are obviously not penalized as heavily as false negatives are.

Overpenalization of near misses: Consider the hypothesized segmentation which assumes a single segment boundary after the first sentence. In this case, 8 errors would be penalized for P_k: 4 when the second pointer of the sliding window points at the second sentence, and another 4 when the first pointer is over the second sentence. P_k would have the double value of the case in the previous

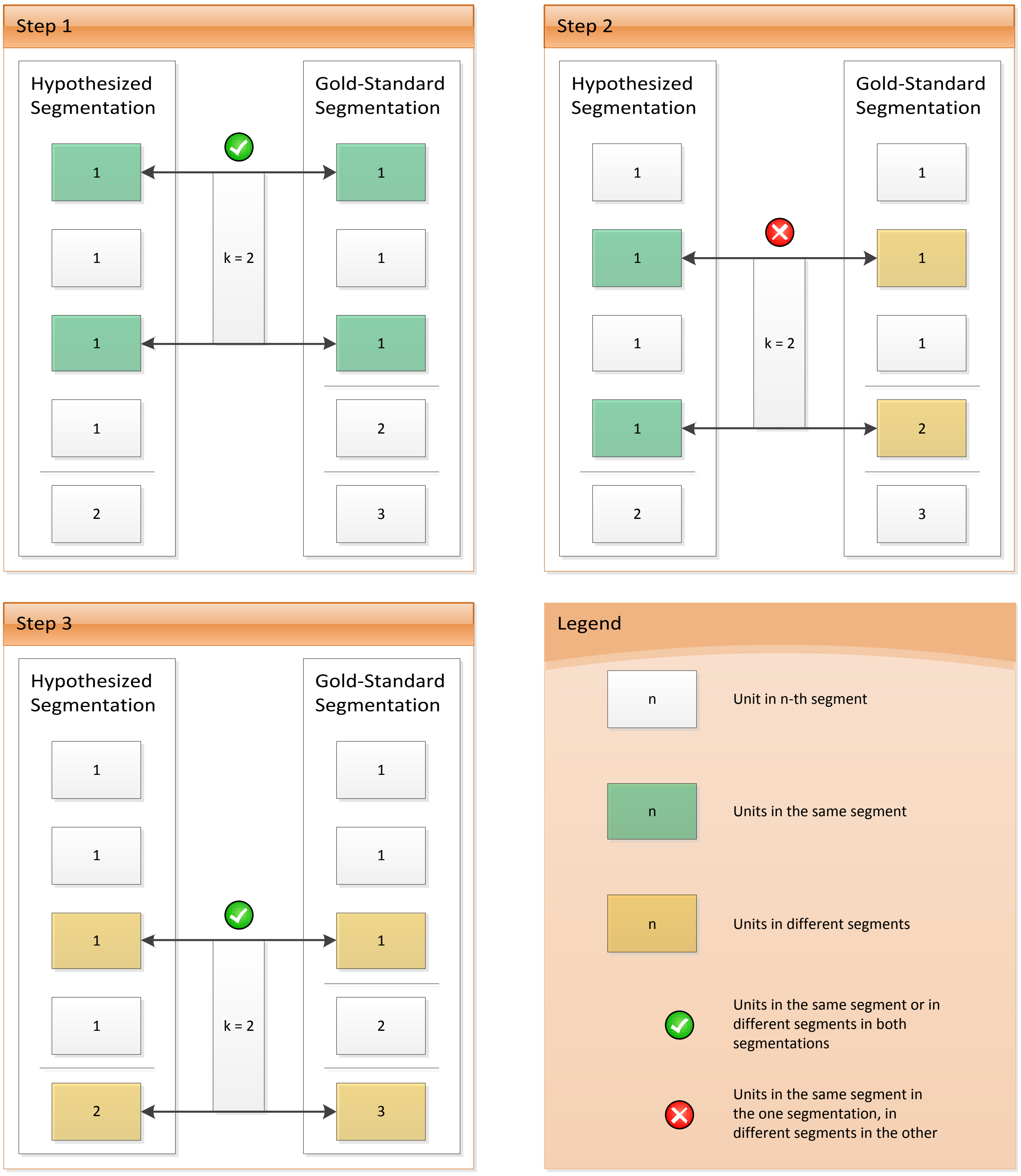

Step 1
Hypothesized Segmentation
Gold-Standard Segmentation
k = 2
1
1
1
1
2
1
1
1
2
3
Step 2
Hypothesized Segmentation
Gold-Standard Segmentation
k = 2
1
1
1
1
2
1
1
1
2
3
Step 3
Hypothesized Segmentation
Gold-Standard Segmentation
k = 2
1
1
1
1
2
1
1
1
2
3
Legend
n
Unit in n-th segment
n
Units in the same segment
n
Units in different segments
Units in the same segment or in different segments in both segmentations
Units in the same segment in the one segmentation, in different segments in the other

Figure 4.1: Calculation of the P_k metric

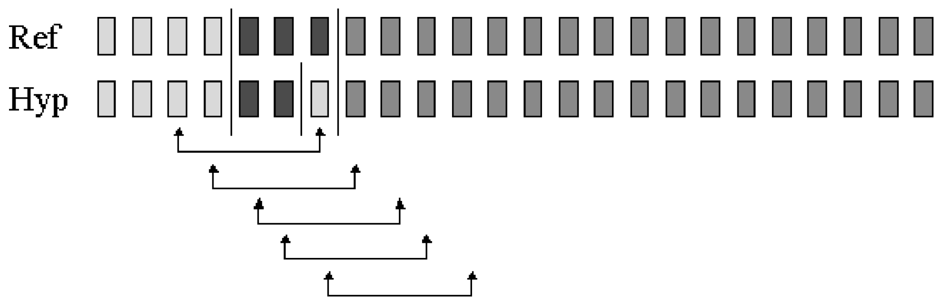


Figure 4.2: Visualization of the problem why P_k may ignore segment boundaries (adopted from Pevzner and Hearst [PH02])

example where two segment boundaries have been hypothesized. This shows that near misses are penalized too much compared to false positives.

Number of segment boundaries ignored: We refer to another example for this case. Think of a reference (Ref) and a hypothesized (Hyp) segmentation as shown in Figure 4.2. The sliding window covers a longer range than contained by the second segment of the gold standard segmentation. If the calculated segmentation wrongly introduces an additionally boundary within this segment as in the figure, no error is counted because at each position of the window, the referenced units are consistently classified ignoring the number of boundaries between the window pointers.

These drawbacks vanish with WindowDiff. WindowDiff is defined exactly as P_k, except that it is more strict in certain cases. Like with P_k, a sliding window moves over the text and, at each position, points at units of the text with a distance of k. Remember the case distinction we applied in the P_k section with respect to the segment membership of the units i and j the window points at: The third case was applied if the units belong to different segments, both in the calculated and in the gold standard segmentation such as in Step 3 of Figure 4.1. This case was never considered to be an error. However, WindowDiff distinguishes two sub cases:

1. The number of segment boundaries between unit i and unit j is *equal* in calculated and gold standard segmentation.
2. The number of segment boundaries between unit i and unit j is *not equal* in calculated and gold standard segmentation.

In WindowDiff, the second sub case is considered to be an error. Formally, WindowDiff is defined very simlarly to P_k:

$$WindowDiff\left(t, Seg_1, Seg_2\right) := \frac{\sum_{i=1}^{|t|-k} error\left(i, i+k, Seg_1, Seg_2\right)}{|t|-k}$$

$$error\left(i, j, Seg_1, Seg_2\right) := \begin{cases} 0 & \text{if } sm\left(j, Seg_1\right) - sm\left(i, Seg_1\right) = sm\left(j, Seg_2\right) - sm\left(i, Seg_2\right) \\ 1 & \text{if } sm\left(j, Seg_1\right) - sm\left(i, Seg_1\right) \neq sm\left(j, Seg_2\right) - sm\left(i, Seg_2\right) \end{cases}$$

$$sm\left(k, Seg\right) := \left|\left\{b \in Seg | b \leq k\right\}\right| + 1$$

Note that only the $error$ function differs from the P_k error function in that it is based on the number of segment boundaries between the two analyzed units, and not only on the membership in the same or in different segments.

The impact on the example in Figure 4.1 is manifested in Step 3 which looks like in Figure 4.3 for WindowDiff. The error in this step is caused by the fact that, between the third and fifth unit, there is

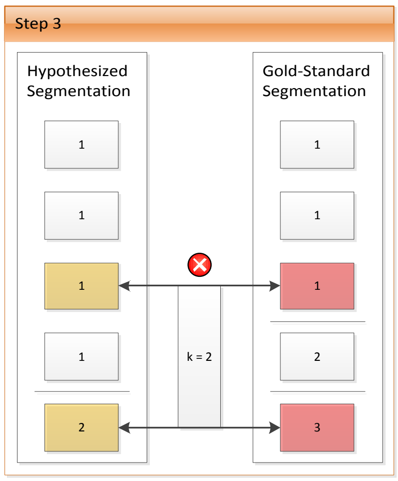


Figure 4.3: Step 3 of the example in Figure 4.1 for WindowDiff

only one segment boundary in the hypothesized segmentation, but there are two in the gold standard segmentation. This is not counted as an error in P_k because the *number* of segment boundaries between the units is not taken into account if it is not zero in both segmentations.

We will now come back to the drawbacks of P_k mentioned at the beginning of this section in order to show that WindowDiff solves most of those problems:

Lax penalization of false positives: Remember the example that a segmentation algorithm, applied to the example text at the beginning of the section, finds one segment boundary after the first sentence and another one after the second sentence. The first one is a false positive, assuming that the latter one is the only boundary in the gold standard. As we have seen, P_k yields a value of 8.5% for this case which is much lower than for a false negative. WindowDiff delivers the same value for the false negative example. In the case of the false positive example, however, WindowDiff counts not only 4, but 15 errors, namely, for each position the sliding window covers the boundary between the first and the second sentence. Thus, the result of WindowDiff in this case is $\frac{15}{47} \approx 31.9\%$ which equals the result for the false negative example. Obviously, with WindowDiff, false positives are no longer rated better than false negatives.

Overpenalization of near misses: If only one segment boundary between the first and the second sentence is hypothesized, P_k and WindowDiff both yield a value of $\frac{8}{47}$ which was, in the P_k case, the double value of what the false positive was rated with. For WindowDiff, the proportion between this value and the false positive rating (31.9%) makes much more sense because a near miss should rather be penalized more leniently than a false positive: A near miss is close to the correct solution whereas a false positive introduces a new wrong boundary.

Number of segment boundaries ignored: Obviously, WindowDiff does not ignore the number of segment boundaries any more. For the segmentations in Figure 4.2, 5 errors are counted at the positions of the sliding window which are displayed in the graphic. P_k does not detect any error in this case.

4.2.3 Tokens or Sentences?

P_k and WindowDiff both depend on the smallest units of documents which are considered for the positions of the sliding window. E.g., for a document with 4 sentences, 80 tokens, and 2 segments, the k parameter of P_k and WindowDiff will be set to either 1 sentence or 20 tokens. In the sentence case, the

Sentence	Segment	# Tokens
1	1	27
2	1	4
3	2	11
4	2	20
$\sum$		62

Table 4.1: Sentence lengths of the example text at the beginning of section 4.2.2

Segmentation	P_k (tokens)	P_k (sentences)	WindowDiff (tokens)	WindowDiff (sentences)
$\emptyset$	31.9%	33.3%	31.9%	33.3%
$\{2\}$	17.0%	66.7%	17.0%	66.7%
$\{3\}$	0.0%	0.0%	0.0%	0.0%
$\{4\}$	46.8%	66.7%	46.8%	66.7%
$\{2,3\}$	8.5%	33.3%	31.9%	33.3%
$\{2,4\}$	31.9%	100.0%	31.9%	100.0%
$\{3,4\}$	23.4%	33.3%	31.9%	33.3%
$\{2,3,4\}$	31.9%	66.7%	63.8%	66.7%

Table 4.2: Comparison of P_k and WindowDiff results for the example text at the beginning of section 4.2.2 on sentence and token base

window iterates over 3 different positions, in case of tokens, 60 positions are analyzed.

The main difference between both variants is that, on token base, length of sentences influences the results of the metrics: For instance, when considering tokens, a near miss which is only a short sentence (e.g., 5 tokens) from the gold standard boundary receives a more lenient penalty than a miss which misses the actual boundary by a long sentence (e.g., 30 tokens). This is why we decided to use tokens as smallest units for the sliding window.

We justify this decision by analyzing P_k and WindowDiff results for the example text at the beginning of Section 4.2.2 on token and on sentence base. The text has sentence lengths as documented in Table 4.1. The gold standard places a segment boundary after the second sentence. Thus, the k parameter of P_k and WindowDiff is set to 1 sentence and 15 tokens, respectively.

Table 4.2 lists P_k and WindowDiff results for given hypothesized segmentations. The first column defines the hypothesized segmentations by enumerating the indexes of sentences before which a segment boundary is placed. As expected, evaluation on token level is more fine-grained, and results on sentence level only comprise the values 0%, 33.3%, 66.7%, and 100% as the maximum number of errors for four sentences and $k = 1$ is 3. Furthermore, as we supposed, evaluation on sentence level over-penalizes near misses with a short distance to the gold standard boundary. This can be seen for the segmentation $\{2\}$ where the short second sentence of the text has wrongly been assigned to the second segment. P_k and WindowDiff only assign a result of 17% on token base whereas, on sentence base, a 66.7% penalty is received which is obviously disproportionately excessive.

The same near miss is contained in the segmentation $\{2,4\}$ which additionally introduces a boundary before the fourth sentence. However, it should not be rated more than twice as bad as the segmentation $\{3,4\}$ which corrects the near miss. On token base, P_k and WindowDiff assign similar, if not equal, results to these segmentations while an evaluation on sentence level yields a result of 100% for $\{2,4\}$ which is, again, not adequate in this case.

In conclusion one can say that evaluation on token level yields more reasonable results, especially when the sentence lengths of the analyzed texts scatter strongly.

4.3 Corpora

An important issue of evaluation is the choice of an appropriate corpus. As defined in Section 4.1, a corpus consists of a set of texts and a gold standard segmentation for each text. Corpora can be divided into two types:

Artificial corpora consist of texts which are each built as a composition of different texts, i.e., every text contains a certain number of coherent parts of different documents which are artificially put together. Segment boundaries are desired at the artificial document part boundaries.

Natural corpora consist of whole texts pertaining to a certain topic each. Segment boundaries are desired between sub topic paragraphs.

While artificial corpora are easier for segmentation algorithms due to the abrupt vocabulary cuts at the boundaries, natural corpora are more realistic since most applications of text segmentations are in the field of natural texts which are to be divided into sub topics.

In order to evaluate the capabilities of our algorithms for both artificial and natural corpora, we chose two corpora for each type, namely the following:

1. Choi's corpus [Cho00] consists of 700 texts. Every text consists of 10 segments. Each segment is the first 3 to 11 sentences of a random document of a subset of the Brown corpus [KF67]. The set can be subdivided into four groups:
   a) 3 to 5 sentences in every segment: 100 texts
   b) 6 to 8 sentences in every segment: 100 texts
   c) 9 to 11 sentences in every segment: 100 texts
   d) 3 to 11 sentences in every segment: 400 texts
2. Galley's TDT[1] and WSJ[2] corpora [GMFLJ03] consist of 500 documents each. Similarly to Choi's corpus, the documents are concatenated artificially with 4 to 22 segments per document. Due to time constraints, our algorithms (Cluster Blocks and Compact Segments) have been evaluated only on 245 documents of the TDT corpus.
3. The corpus used by Malioutov and Barzilay [MB06] comprises transcripts of university lectures. 22 are taken from a lecture about artificial intelligence (AI), 33 are about physics. AI lectures have been transcripted manually, the physics lectures have been transcripted both automatically and manually. As the quality of automatic transcription in this case is very low, we only consider the manual transcripts. Thus, this corpus contains 55 documents.
4. As a further corpus, we designed a corpus extracted from the English Wikipedia which contains 57 documents. For a detailled description of this corpus, refer to the next section.

The first two corpora are obviously artificial, the latter two are natural.

[1] Topic Detection and Tracking
[2] Wall Street Journal

Name	Type	d	t/d	σ (t/d)	t/sen	seg/d	σ (seg/d)
Choi	artificial	700	1985.8	504.6	25.59	10	0
Galley (TDT)	artificial	500	4277.7	1996.6	30.18	13	5.74
Galley (WSJ)	artificial	500	7221.3	4596.2	23.89	13	5.74
Malioutov	natural	55	8540.4	1410	17.65	8.42	4.17
Wikipedia	natural	57	4626.3	2028.9	26.28	6.7	2.28

Table 4.3: Characteristics of evaluation corpora: "d" contains the number of documents in the corpus. "t/d" denotes the average number of tokens per document. "t/sen" is the average number of tokens per sentence. "seg/d" is the average number of segments per document. The "σ" columns indicate standard deviations.

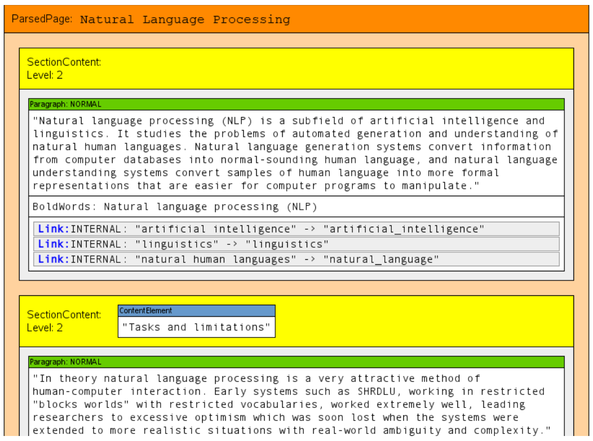

```
ParsedPage: Natural Language Processing
SectionContent:
Level: 2
Paragraph: NORMAL
"Natural language processing (NLP) is a subfield of artificial intelligence and
linguistics. It studies the problems of automated generation and understanding of
natural human languages. Natural language generation systems convert information
from computer databases into normal-sounding human language, and natural language
understanding systems convert samples of human language into more formal
representations that are easier for computer programs to manipulate."
BoldWords: Natural language processing (NLP)
Link:INTERNAL: "artificial intelligence" -> "artificial_intelligence"
Link:INTERNAL: "linguistics" -> "linguistics"
Link:INTERNAL: "natural human languages" -> "natural_language"
SectionContent:
Level: 2
ContentElement
"Tasks and limitations"
Paragraph: NORMAL
"In theory natural language processing is a very attractive method of
human-computer interaction. Early systems such as SHRDLU, working in restricted
"blocks worlds" with restricted vocabularies, worked extremely well, leading
researchers to excessive optimism which was soon lost when the systems were
extended to more realistic situations with real-world ambiguity and complexity."
```

Figure 4.4: Structure of a Wikipedia article (adopted from Zesch et al. [ZMG08])

An overview of the corpora we used for evaluation is given in Table 4.3. Salient is that the number of tokens in the Choi documents is rather small compared to other corpora. Also, we see that Choi's corpus is the only one using a fixed number of segments for every document. On the one hand, this facilitates tuning of segmentation algorithms for this corpus; on the other hand, it is not realistic and may lead to over-fitting during tuning so that the tuned algorithms will always find a segment number close to 10. The number of tokens per sentence is low in Malioutov's corpus which has its reason in the spoken-speech nature of this corpus. This might make it difficult for algorithms to find exactly correct boundaries for this corpus. However, since we use tokens and not sentences as smallest unit during evaluation (see Section 4.2.3), near misses are penalized leniently and, thus, the representative character of results for this corpus should not suffer from it.

4.3.1 Wikipedia Corpus

Natural-text corpora are difficult to generate because reasonable gold standard segment boundaries for a big collection of texts are rare. Creating a gold standard manually would be too laborious and time-consuming for this thesis. Thus, we decided to extract a corpus from articles of the English Wikipedia[3] and to take the division into sections as a gold standard for segments.

Figure 4.4 shows an example structure of a Wikipedia article. For corpus generation, we extracted top-level sections of articles and concatenated their textual contents to a pure-text corpus file. The con-

[3] http://en.wikipedia.org/

```
==========
Sentence 1.1
Sentence 1.2
...
Sentence 1.S_1
==========
Sentence 2.1
...
==========
...
==========
...
Sentence N.S_N
==========
```

Figure 4.5: Format of Choi's corpus. N is the number of segments in the document. For each segment n, S_n is the number of sentences in n.

tent of a section is constituted by the concatenation of the text of its paragraph elements and the content of contained sections. Particularly, other elements such as tables and image captions are ignored during generating the text for a section because text segmentation is meant to be applied to prose and not to pieces of information such as table fields. Furthermore, sections with one of the titles "See also", "References", and "External links" are skipped as they do not contain information where segmentation makes sense.

As Wikipedia is a free-to-all knowledge resource where everybody can edit articles, many articles are quite short, erroneous, or exhibit a sloppy – or even missing – division into sections. However, there is a category of articles called "Featured Articles" which only contains well elaborated texts. In particular, separation into sections is in almost all cases appropriate to reasonably represent sub topics of the text and, thus, to be used as gold standard segments for the corpus. Hence, we filtered out Featured Articles from the English Wikipedia to use them for the Wikipedia corpus. Out of 1244 existing Featured Articles, we randomly selected a set of 57 for our corpus.

4.3.2 Format

This section deals with the format in which evaluation corpora are serialized. We will first look at the format Choi defined for his corpus and which has been used in several other works such as for Malioutov's and Galley's corpora. Afterwards, we describe and justify the format we have used for our Wikipedia corpus and we have converted the other corpora to.

Choi defines the format as presented in Figure 4.5: Every document begins and ends with the delimiting character sequence "==========" which is also inserted at every gold standard segment boundary. Every sentence within a segment begins a new row.

We suggest a separated format instead: Text and segment boundaries are separated into two files. The text file simply contains the sentences of the text, without any line breaks as they actually provide information to the segmenter about sentence boundaries which would not be available in real systems. The gold standard file contains one line for each character offset of a segment boundary. E.g., for a boundary after the 200st character of the text file, the gold standard file would contain a line with the number “200”. The numbers refer exactly to the positions of the boundary in the text file. Particularly, if the text file is encoded in Windows-style, i.e., with line feed and carriage return at each line end, every line break is accordingly counted as two characters. However, if a character in the text file takes two bytes, e.g., in UTF-16 encoding, it is, nevertheless, counted as one single character.

The separated format has three advantages compared to Choi’s format:

- It does not provide more information to the segmenter than available in a real application. Particularly, sentence breaks are not provided.
- Text and boundaries are separated. This means, for one text, one could theoretically provide a set of gold standards, possibly with different levels of granularity, without redundancy.
- The format allows boundaries within sentences. This might be useful for evaluation of fine-grained segmenters which are to discover topical shifts also within sentences.

4.4 Tuning

The Cluster Blocks and Compact Segments algorithms have different parameters that must be tuned for optimum results. Admittedly, optimum parameter values may depend on the kind of texts which the algorithms are applied to: E.g., for longer texts, parameters causing a more coarse-grained segmentation are possibly more appropriate than for shorter texts. However, this is a drawback most unsupervised algorithms cannot avoid.

In this section, we give an overview of the parameters we have tuned for both algorithms and describe the methodology.

4.4.1 Tuning Corpora

We decided to optimize the tuning parameters on subsets of Choi’s corpus because this artificial corpus – where each documents consists of concatenated parts of different texts – is appropriate to analyze whether an algorithm detects obvious and unique boundaries. As a first tuning step, we optimized the parameters on a subset of five randomly chosen documents. In a second step, we compared results of promising configurations on a corpus of 50 randomly selected documents. Finally, the best configuration was chosen for evaluation on the whole corpus and on other corpora.

4.4.2 Parameters

For the Cluster Blocks algorithm, tuning was performed for the following parameters[4]:

Sparsification mode (SM): This parameter has one of the values *Threshold* and *Quota* and defines how edges are removed from the complete semantic graph before clustering. For details, refer to Section 3.1.3.

Sparsification value (SV): Defines either the threshold value for edges or the quota of edges which should be removed. This is a value between 0 and 1. For tuning, we chose the domain $\{0, 0.1, \ldots, 1\}$.

Limit for number of edges (SL): Defines the maximum number of edges to be retained in the sparsification process. If there are more edges remaining after applying the threshold or quota, excess edges

[4] Letters in brackets define abbreviations for the parameters which are used in tables later.

with low semantic relatedness values are removed. This parameter is necessary due to runtime reasons, because for long documents, even a high threshold may yield many edges leading to a long runtime since, when ignoring this upper limit for the number of edges, runtime complexity is in $O\left(T^5\right)$ where T is the number of terms in the document as we have seen in Section 3.1.7.

Smoothing mask for clustering quality values (CMM): This tuple defines the mask with which modularity values of different possible clusterings are smoothed. Details are described in step 6 of Section 3.1.4. As domain for this parameter, we chose the masks (1)[5], $(1,1,1), (1,2,1), (1,5,1), (1,2,4,2,1)$.

Maximum number of sequential sentences in a block without a word of the cluster (MS): If a block would contain a greater gap than the number defined with this parameter, it is split into multiple blocks. The domain is $\{0,1,2,3,4\}$.

Minimum length of a block (MBL): Blocks with less sentences than defined with this parameter are not allowed. The domain is $\{1,2,3,4\}$.

Minimum length of a segment (MSL): This floating point value denotes the minimum length a segment must have relative to the average segment length. If a boundary causes a segment to be too small, it is discarded. We chose the domain $\{0,0.2,0.4,0.6,0.8,1\}$.

Smoothing mask for boundary strength values (BSM): A tuple of values defining the mask with which boundary strength values are smoothed. Details can be found in Section 3.1.6. This parameter has the same domain as CMM.

For the Compact Segments algorithm, tuning was done according to equation (3.1). Hence, the parameters in this case are:

Weight of compactness (α): A value between 0 and 1 where 1 denotes highest weight for compactness and 0 denotes highest weight for boundary strength. We chose the domain $\{0,0.1,\ldots,1\}$.

Length recompense factors (β,γ) for compactness and boundary strength, respectively. We tested values of the domain $\{0,0.1,\ldots,0.9,1,1.2,1.5\}$.

Segment penalizer (d): Higher values for this parameter will cause the number of segments to be lower. Therefore, we did not restrict its domain (except that we did not take negative values into account because the value 0 already yields the maximum number of segments in every case) in order to be able to tune it as necessary in every step.

Adjacent sentences (S_{adj}): This additional parameter denotes the number of sentences which are considered to be "adjacent" for a certain segment candidate in the text. This influences calculation of boundary strength (see Section 3.2.4). We tested values from 1 up to 5.

4.4.3 Tuning Results

We performed tuning of the Cluster Blocks algorithm with provided number of segments as automatic detection is, in this algorithm, not reliable enough. We tested 73 promising configurations on the five documents of the Choi tuning corpus. This revealed the configurations shown in Table 4.4 to yield the best results on the tuning corpus. The configuration in italic font is used for further evaluations.

Obviously, sparsification should be done in "Quota" mode and not in "Threshold" mode. This is due to the fact that distribution of semantic relatedness values strongly differs between different documents:

[5] No smoothing at all.

SM	SL	SV	CMM	MS	MBL	MSL	BSM	P$_k$	WindowDiff
Quota	*1000*	*0.7*	*(1, 2, 1)*	*3*	*3*	*0.4*	*(1, 5, 1)*	*18.8%*	*19.4%*
Quota	1000	0.7	(1, 2, 1)	2	3	0.4	(1)	20.2%	20.7%
Quota	1000	0.7	(1, 2, 1)	2	3	0.4	(1, 5, 1)	21.4%	22.0%
Quota		1	(1, 2, 1)	3	3	0.4	(1, 5, 1)	28.8%	29.2%

Table 4.4: Best configurations of the Cluster Blocks algorithm for the tuning corpus. The last configuration is the best one abandoning semantic data, i.e., basing on word reiterations only.

α	β	γ	S_{adj}	P$_k$	WindowDiff
0.9	1	0.6	4	12.4%	12.8%
0.5	1	0.6	4	12.6%	13.1%
0		0.6	4	12.6%	13.1%

Table 4.5: Best configurations of the Compact Segments algorithm for the tuning corpus with given number of segments

While, in one document, a value of 0.3 is relatively high, in another document this may be too low to respect in the sparsificated semantic graph. The quota mode always retains the same quota of semantic relations which yields best results across multiple documents.

The upper limit of 1000 semantic relations has been chosen due to runtime reasons: Although a higher limit might improve results on large documents, this would impair runtime performance disproportionately compared to the degree of improvements.

The value SV = 0.7 indicates that ca. 30% of the strongest edges should be retained in the sparsificated graph. The mask $(1, 2, 1)$ for smoothing the clustering quality values shows that smoothing indeed makes sense because weak smoothing ($(1, 5, 1)$) or no smoothing ((1)) yielded significantly worse results. The importance of smoothing the boundary strength values, on the contrary, did not prove to be very high since weak smoothing or no smoothing yielded best results.

The last row of table 4.4 shows the same configuration as the best one in the first row, except for SV = 1, i.e., all semantic relations are removed and blocks in the text are detected based on pure word reiterations. Results are significantly worse than for the first configuration which suggests that usage of the semantic graph indeed enhances finding semantically related blocks.

The best configuration of Table 4.4 yielded P$_k$ and WindowDiff values of 24.1 and 27.6%, respectively, if the number of segments was not provided to the algorithm. This is also the configuration we chose for further evaluations.

For given numbers of segments, the best results of Compact Segments on the tuning corpus could be achieved with the configurations presented in Table 4.5.

Interestingly, the value of α which weights compactness against boundary strength does not seem to have much impact on the quality of the found segmentation. However, if we chose α to be even greater than 0.9 (e.g., 0.95 or 1), results became significantly worse. This indicates that compactness only is not sufficient to find a good segmentation. However, as $\alpha = 0$ still yields good results, boundary strength alone seems to be a sufficient indicator for good segments. Yet, best results could be achieved with $\alpha = 0.9$ which shows that a combination of compactness and boundary strength should be the best choice. But as mentioned, segmentations are quickly getting instable for greater values of α whereas for

α	β	γ	d	S_{adj}	P$_k$	WindowDiff
0.5	*1*	*0.6*	*0.05*	*4*	*11.2%*	*11.6%*
0.9	1	0.6	0.009	4	11.5%	13.1%
0.9	1	0.6	0.01	4	13.0%	13.3%
0.5	1	0.6	0.045	4	11.7%	13.4%

Table 4.6: Best configurations of the Compact Segments algorithm for the tuning corpus without given number of segments

Corpus	**No**	**All**	**C99**	**C99$_b$**	**TT**	**MinCut**	**U00**	**LCseg**	**CB**	**CS**
Choi	46.8%	52.9%	12.7%	14.9%	49.9%	21.2%	**10.5%**	**10.5%**	33.3%	12.1%
Malioutov	42.3%	57.7%	38.6%	53.5%	57.0%	**32.8%**	34.2%		57.4%	42.0%
Galley (TDT)	43.9%	56.0%	10.2%	15.7%	51.9%		8.7%	**7.0%**	28.7%	19.0%
Galley (WSJ)	36.1%	63.9%	22.3%	24.1%	57.3%		17.7%	**15.3%**		
Wikipedia (57)	**37.6%**	62.4%		50.3%	59.3%				59.5%	50.4%

Table 4.7: Collected P$_k$ results of different algorithms on different corpora

all values between 0 and 0.9, the same segmentations were found for the tuning documents; hence, we decided to choose the balance of compactness and boundary strength ($\alpha = 0.5$).

The values of δ are not listed in the table because it does not influence the chosen segmentations if the number of segments is provided. If the number was not provided, the configurations in Table 4.6 turned out to be the best ones for the tuning corpus. The configuration in italic font is used for further evaluations.

In this case, $\alpha = 0.5$ was most reliable for finding a good number of segments which confirms our decision not to choose $\alpha = 0.9$: With $\alpha = 0.9$, d values must be chosen very small and insignificant in order to obtain a reasonable number of segments. This suggests an instable behavior of the algorithm. For $\alpha = 0.5$, we can choose a slightly greater value for d (0.05). Thus, we chose the first configuration listed in the table for further evaluations.

4.5 Results

Table 4.7 shows collected P$_k$ results of different algorithms on the corpora which have been introduced in Section 4.3. Unfortunately, we did not have the opportunity to perform evaluation of all algorithms ourselves; in these cases, we fell back on the best values we found in available papers. Empty cells indicate that results could not be found or calculated.

The following algorithms are considered in the table:

No: Baseline algorithm which does not set any segment boundary at all; i.e., the whole text is one big segment in the output of this algorithm.

All: Baseline algorithm which sets a segment boundary wherever possible, i.e., between all sentences.

C99: This column contains values we have found in available papers on Choi's C99 [Cho00]. Values stem from Choi himself, Galley et al. [GMFLJ03], and Malioutov and Barzilay [MB06].

C99$_b$: Choi's C99 component we have used for evaluation. This component has not been tuned, e.g., with respect to the mask size. Instead, the default configuration (11×11-matrix) has been applied to every corpus.

Corpus	No	All	C99	C99$_b$	TT	MinCut	U00	LCseg	CB	CS
Choi	46.8%	98.2%	14.6%	17.6%	65.9%	23.4%	11.6%	**11.4%**	36.8%	12.8%
Malioutov	42.3%	100.0%	40.5%	79.3%	97.0%	**34.8%**	36.3%		96.8%	44.9%
Galley (TDT)	43.9%	99.5%	12.7%	19.3%	77.6%		11.1%	**9.1%**	33.8%	21.2%
Galley (WSJ)	36.1%	100.0%	29.8%	32.1%	83.1%		24.1%	**22.1%**		
Wikipedia (57)	**37.6%**	100.0%		58.1%	88.2%				83.6%	55.2%

Table 4.8: Collected WindowDiff results of different algorithms on different corpora

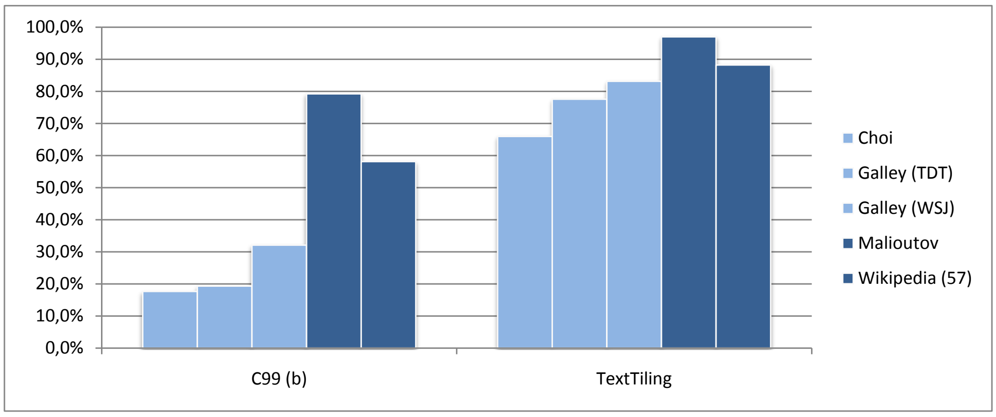


Figure 4.6: WindowDiff results of C99 and TextTiling on artificial (bright) and natural (dark) corpora

TT: Choi's implementation [Cho99] of TextTiling [Hea94] we have used for evaluation. This component has not been tuned, either.

MinCut: Minimum Cut Model of Malioutov and Barzilay [MB06]. Vaues stem from the same paper.

U00: Utiyama's and Isahara's probabilistic U00 algorithm [UI01]. Values stem from Galley et al. [GMFLJ03] and Malioutov and Barzilay [MB06].

LCseg: Galley's LCseg algorithm [GMFLJ03]. Values stem from the same paper.

CB: Our Cluster Blocks algorithm in the configuration presented in the previous section.

CS: Our Compact Segments algorithm in the configuration presented in the previous section.

Table 4.8 presents analogous values for the WindowDiff measure.

When comparing the results across different corpora, observe that artificial corpora (Choi, Galley) yield clearly better results than natural corpora (Malioutov, Wikipedia). It is illustrated for C99 and TextTiling in Figure 4.6 where bars of artificial corpora have bright color, bars of natural corpora have dark color. This confirms the proposition formulated in Section 4.3 that artificially concatenated documents are much easier to segment due to abrupt vocabulary changes at segment boundaries. However, this raises the question whether P_k and WindowDiff measures are fair for evaluation of segmentations on natural documents. They are claimed to be comparable across different types of documents which is obviously not fulfilled when artificial and natural documents are used. We think that the reason for it is the concept of P_k and WindowDiff which permits only one "correct" segmentation for each document. This is appropriate for artificial documents where segment boundaries are objectively unique, but not for

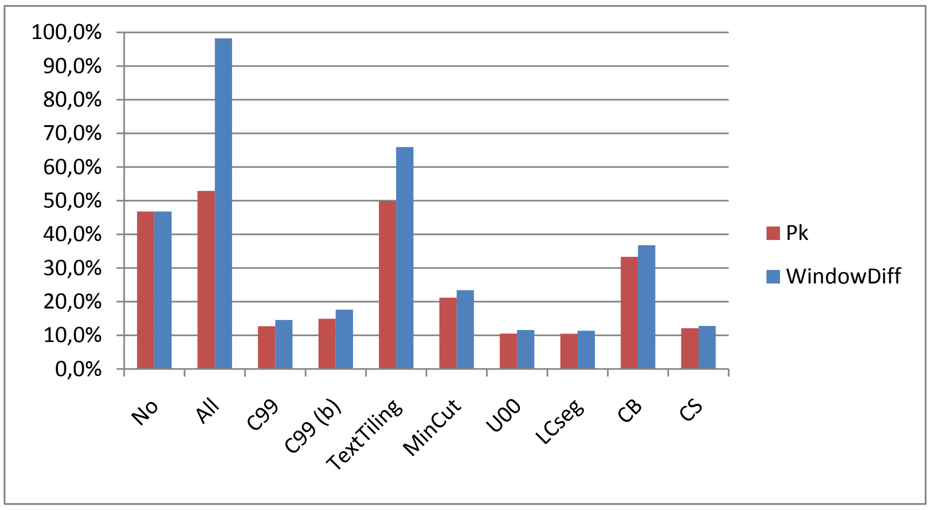


Figure 4.7: P_k and WindowDiff results of different algorithms on Choi's corpus

Corpus	**Eq**	**C99**	**C99$_b$**	**U00**	**LCseg**	**CB**	**CS**
Choi	41.4%	11.1%	14.1%	8.8%	**8.7%**	32.0%	10.8%
Malioutov	49.2%		**41.1%**			47.6%	43.4%
Galley (TDT)	46.8%	9.4%	15.1%	**4.7%**	6.2%	24.8%	14.2%
Galley (WSJ)	47.7%	19.6%	20.5%	15.2%	**12.2%**		
Wikipedia (57)	52.3%		**42.7%**			47.3%	44.4%

Table 4.9: Collected P_k results of different algorithms on different corpora when the gold standard number of segments is provided

natural documents where even different human annotators would likely choose different segments.[6]

The difficulty of segmenting natural texts can also be read from the fact that the baseline algorithm "No" which does not place any segment boundary achieves best results (37.6% P_k and WindowDiff) on the Wikipedia corpus. It would be interesting to see whether high-end algorithms such as U00, LCseg, or also MinCut which has been designed for segmentation of natural texts perform better on Wikipedia. For Malioutov's natural corpus, at least, MinCut and U00 outperform the "No" baseline significantly.

Figure 4.7 compiles P_k and WindowDiff results of different algorithms on Choi's corpus. Analyzing the results of the algorithms, we see that U00 and LCseg obviously belong to the most reliable segmentation algorithms – also across different corpora as can be seen in the table –, although also results of C99 are still respectable. Cluster Blocks, with $P_{k_\oslash}(\text{CB}, \text{Choi}) = 33.3\%$ and $\text{WindowDiff}_\oslash(\text{CB}, \text{Choi}) = 36.8\%$, cannot compete with state-of-the-art algorithms; however, it is still better than the baseline algorithms and TextTiling. Compact Segments with P_k of 12.1% and WindowDiff 12.8% performs slightly better than Choi's C99 (12.7 and 14.6%); however, it cannot achieve values of U00 and LCseg. On Choi's corpus, Compact Segments has also been tested without stop word removal which yielded P_k and WindowDiff values 13.6% and 14.3%, respectively. As expected, with an improval of 1.5%, stop word removal contributes to better detection of semantic structures and, thus, to better segmentation results.

[6] Yet, understand this as a remark beyond the scope of this thesis; we will nevertheless continue using those metrics for evaluation. However, at this point, we want to allude to an analysis of WindowDiff by Lamprier et al. [LALS07] who point out similar disadvantages and suggest alternatives which might be suited to supersede WindowDiff in the future.

Corpus	Eq	C99	C99$_b$	U00	LCseg	CB	CS
Choi	42.0%	13.9%	15.7%	**9.4%**	**9.4%**	33.6%	11.8%
Malioutov	50.1%		**43.5%**			48.9%	46.5%
Galley (TDT)	48.5%	11.9%	18.4%	**6.3%**	8.4%	27.3%	16.6%
Galley (WSJ)	51.9%	26.4%	27.3%	21.5%	**18.3%**		
Wikipedia (57)	52.9%		**45.1%**			48.5%	45.9%

Table 4.10: Collected WindowDiff results of different algorithms on different corpora when the gold standard number of segments is provided

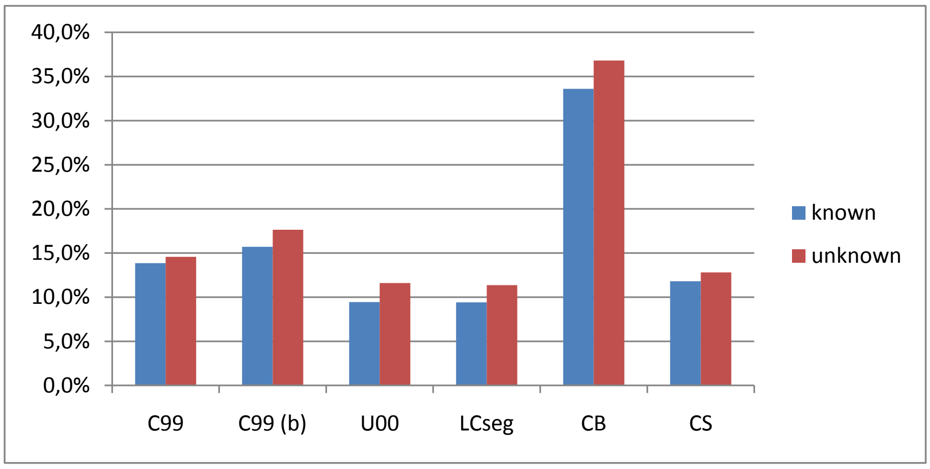


Figure 4.8: WindowDiff results of different algorithms on Choi's corpus with and without provided number of segments

Besides, we have also collected P$_k$ and WindowDiff results for some algorithms where the number of segments in the gold standard has been provided. Data can be found in Tables 4.9 and 4.10. Note that, in this case, the algorithms "No" and "All" have been replaced by "Eq", another baseline algorithm, placing the correct number of segment boundaries at equidistant positions. All results of C99, U00, and LCseg have been adopted from the work of Galley et al. [GMFLJ03].

Figure 4.8 juxtaposes WindowDiff results for known and unknown number of segments on Choi's corpus. Absolute improvement for state-of-the-art algorithms is up to 2%.

Figure 4.9 shows much stronger improvement for the natural corpus Wikipedia which indicates that the number of segments is difficult to compute for natural texts which is not very astonishing since segmentations of natural texts may be more or less fine-grained. The problem is that – as mentioned before – current evaluation metrics only allow for one single gold standard.

The Cluster Blocks algorithm sticks out with its improvement from 83.6% WindowDiff before to 48.5% with provided number of segments. For Malioutov's corpus, improvent is even more salient. We suggest this to be due to the way in which the number of segments is determined in the algorithm: A threshold (mean value plus standard deviation) is used for the boundary strength, and sentence gaps with higher values are chosen as segment boundaries. In natural texts, there are often not very salient boundary strength values; thus, standard deviation is low and many values are above the threshold which may lead to an exaggerated number of segments.

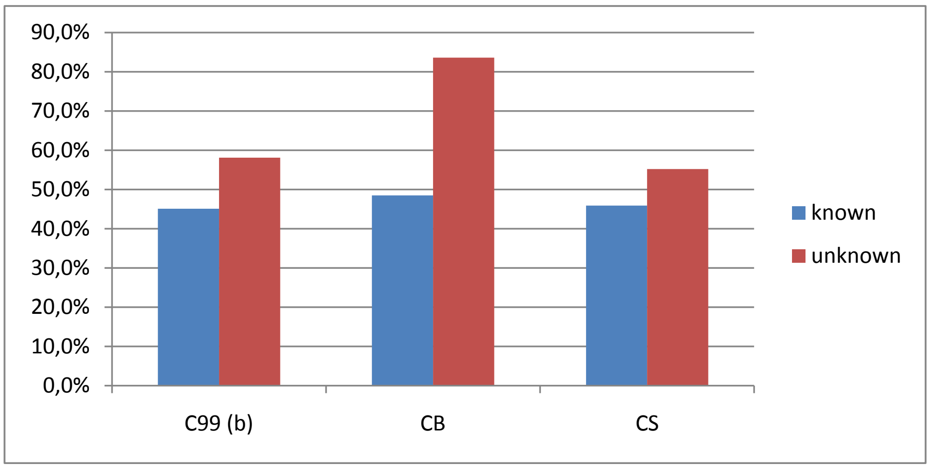


Figure 4.9: WindowDiff results of different algorithms on the Wikipedia corpus with and without provided number of segments

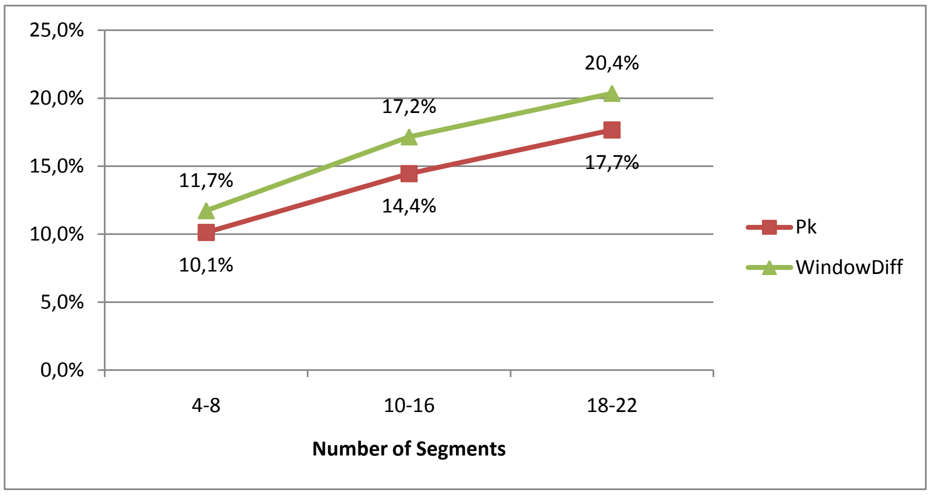


Figure 4.10: WindowDiff results of Compact Segments with provided number of segments for the TDT corpus

An interesting observation concerning the Compact Segments algorithm can be made in Figure 4.10 which shows WindowDiff results for three categories of documents within Galley's TDT corpus. Every category contains only those documents whose number of segments matches the range given by the category's name. Obviously, Compact Segments results become worse for a greater number of segments which we have not observed in that extent for other algorithms. Yet, detailed analysis of this problem is left for future work.

5 Implementation

The previously presented algorithms *Cluster Blocks* and *Compact Segments* have been implemented in IBM's Natural Language Processing Framework UIMA[1]. This chapter shortly imparts a basic understanding of the UIMA concepts and then describes the UIMA pipelines and components that have been used and implemented in the course of the Bachelor Thesis.

Refer to Section 5.8 if you are searching for a compact overview of all implemented UIMA components.

5.1 UIMA

UIMA is a framework for management of unstructured information such as images, audio data, or – most frequently used – text. The UIMA project was started in 2005 by IBM[2] and is now supervised by the Apache Software Foundation[3]. It is freely available on the Apache website and can be used with the programming languages Java and C++.

UIMA is based on a pipeline concept: It considers every document as an artefact which moves through a pipeline of components. Every component analyzes the document as needed and has access to a central pool of document annotations where analyzation results may be saved or previous results may be received from. All document-specific data is saved in a CAS[4] which contains all document contents and the annotations that have been added.

Components are generally divided into three kinds of types:

Collection Readers are placed at the beginning of a pipeline and load the documents to be analyzed, e.g., from a certain input directory. The document text is saved in the UIMA structure and will not be changed throughout the pipeline.

Annotators analyze the loaded documents and read and/or add annotations to the document which may pertain to the whole document or even only to a certain part of the document.

Consumers read the document and its annotations and process them in some way. For instance, all data might be saved to a file.

Frequently needed components such as file readers/writers, sentence or token splitters etc. are already provided although implementation of new components is easy to handle. This simple and effective structure of UIMA has made it become the standard framework in NLP research and industry.

Within this Bachelor Thesis, we did not only use the standard UIMA components, but also components of DKPro[5], a collection of advanced flexible UIMA components [MZM+08].

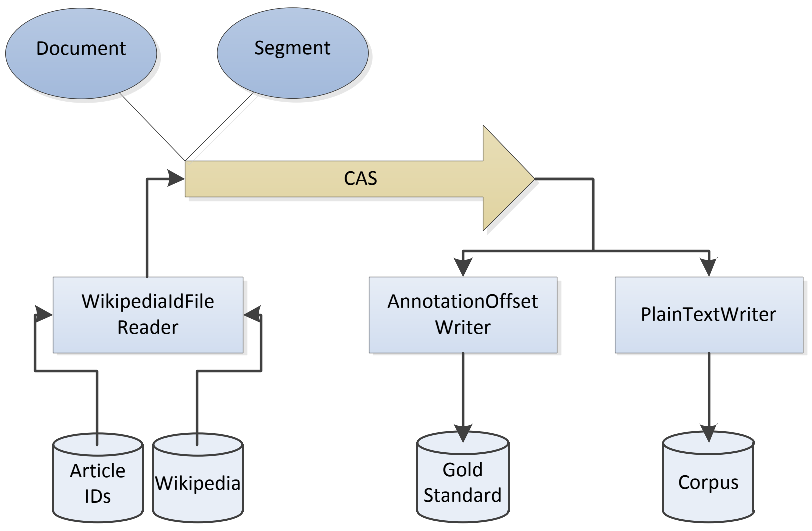


Figure 5.1: UIMA pipeline for generation of the Wikipedia corpus

5.2 Corpus Generation

For generation of our Wikipedia corpus (see Section 4.3.1), we used the pipeline presented in Figure 5.1. The components are described below.

WikipediaIdFileReader*: This component bases on JWPL[6], a freely available Wikipedia API of Zesch et al. [ZMG08] which enables easy access to Wikipedia articles and categories. The WikipediaIdFileReader reads a file expecting it to contain numeric IDs of Wikipedia articles to be read in. In our case, we used a list of IDs of featured articles. The articles are received and parsed with JWPL and Segment annotations are added according to the article's structure (see the PARAM_SECTION_GRANULARITY parameter for details). The text which is finally set as document text does not contain table elements, image captions or the like, but only text passages. The configuration parameters are set as following:

HOST, DB, USER, PASSWORD define the database connection which provides access to Wikipedia articles.

LANGUAGE = Language.english

PARAM_SECTION_GRANULARITY = 1
This integer value defines the section level on which to split articles into segments. I.e., the value 1 means that only top-level sections are separated; the value 2 would stand for a segmentation on the second level: E.g., if an article has three top-level sections which are each divided into two sub-sections, that configuration would yield six segments for this article.

PARAM_SPLIT_PARAGRAPHS = false
This defines whether different paragraphs at a higher level than defined with PARAM_SECTION_GRANULARITY should constitute separate segments. E.g., if granularity is set to 2, multiple sequenced paragraphs on the first section level would be put into different segments if this parameter was true.

[1] Unstructured Information Management Architecture
[2] http://www.research.ibm.com/uima/
[3] http://uima.apache.org/
[4] Common Analysis Structure
[5] Darmstadt Knowledge Processing Repository
[6] Java-based WikiPedia Library

PARAM_SEGMENT_DELIMITER = " " (space character)
: This parameters defines the string which separates two segments. We use only a space character, thus, not separating segments into paragraphs in document texts.

AnnotationOffsetWriter*: Writes the character offsets of a certain annotation type to a text file. In our case, we apply this component to the annotation type Segment in order to save the gold standard segmentation. The index of the first annotation is not saved as it is assumed that the first annotation always begins at the first character. E.g., if the document contains three sections with character offsets 0 - 120, 121 - 278, and 279 - 440 (all including), the component would only write the two numbers 121 and 279 to the file as the other data can be restored from them and from the document length. The configuration is:

PARAM_ANNOTATION_TYPE = de.tudarmstadt.ukp.dkpro.semantics.type.Segment

PARAM_OUTPUT_DIRECTORY = *output directory*
: Defines the directory where to save the text file with the segment boundary offsets. The name of the file is constituted by the document meta data, particularly, the document URI and its title.

PARAM_PATH_LEVELS_TO_CARRY_OVER_FROM_INPUT_TO_OUTPUT_FILE = 0
: In this pipeline, the value is not needed. Generally, the value constitutes how many parts of the input file path should be copied to the path of output files. E.g., if input files are read recursively from the directory "dirA/dirB/dirC" and are all located in one of the sub directories "dir1", "dir2", and "dir3", the value must be set to 1 if the structure of the result files should be equal, i.e., if gold standard segmentation files for input files from "dir1" are to be located in the sub directory "dir1" of the output directory. If all gold standard files are to be located in the same directory and not in sub directories, this value must be 0.

PlainTextWriter*: Writes the document texts without any annotations to text files. The parameters PARAM_OUTPUT_DIRECTORY and PARAM_PATH_LEVELS_TO_CARRY_OVER_FROM_INPUT_TO_OUTPUT_FILE are defined as for the AnnotationOffsetWriter.

Components with an asterisk (*) have been implemented in the course of this thesis.

5.3 Preprocessing

The preprocessing pipeline is shown in Figure 5.2. Rectangles below the CAS pipeline arrow constitute UIMA components. Arrows from the pipeline to components indicate reading annotations, arrows to the pipeline indicate writing annotations. Added annotation types are represented as ellipses above the pipeline arrow.

This pipeline consists of DKPro components which are, together with their configuration, shortly described in the following list:

FileSystemReader: Reads all text files of a certain directory. Saves document meta data such as the file URI to the annotation pool.

UkpSentenceSplitter: Divides the text into sentences by adding according Sentence annotations.

UkpTokenizer: Adds Token annotations for words in the text which are separated by punctuation marks or spaces.

TreeTaggerPosLemmaTT4J: Finds lemmatized roots of tokens and adds corresponding Lemma annotations. Also annotates the tokens with part-of-speech tags.

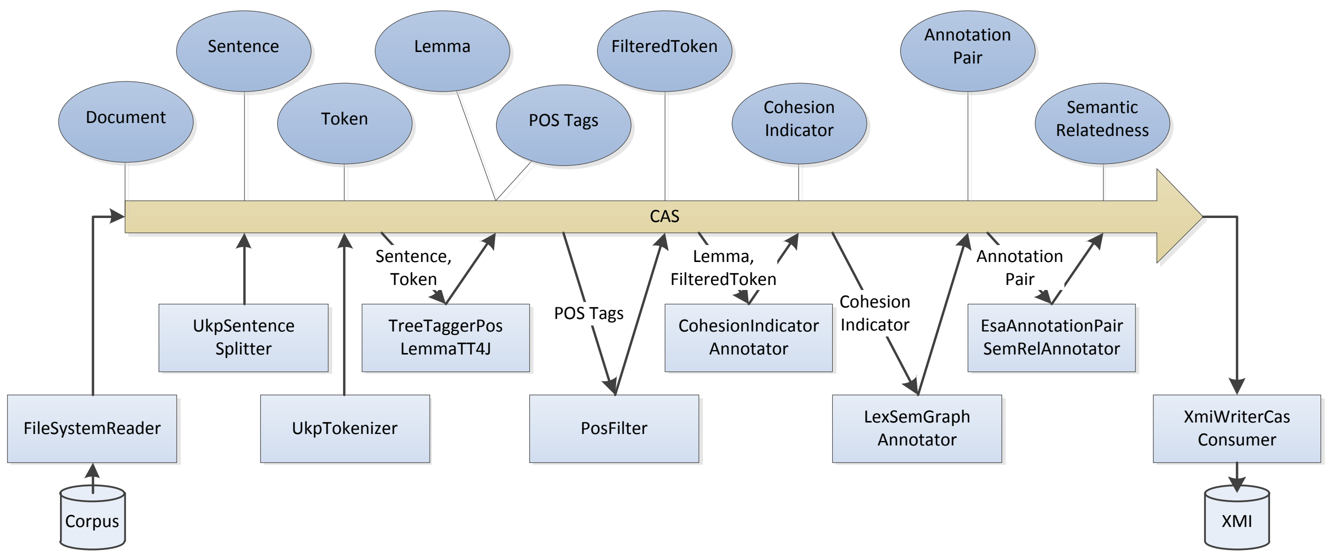


Figure 5.2: UIMA preprocessing pipeline

PosFilter: Selects certain parts of speech by annotating them with FilteredToken annotations. The parameters in our configuration are:

PARAM_ADJ = true

PARAM_N = true

PARAM_V = true

This selects adjectives, nouns, and verbs.

CohesionIndicatorAnnotator*: Selects tokens which are to be used for segmentation analysis. For that purpose, Lemmas are tagged with a CohesionIndicator annotation. As we only want to use those lemmas which are annotated as FilteredTokens, the parameter configuration is:

PARAM_USE_POS_FILTER = true

LexSemGraphAnnotator: Creates an AnnotationPair for every pair of CohesionIndicators in the text. This actually constitutes the edges of the semantic graph. The parameter configuration is:

PARAM_ANNOTATION_TYPE = "de.tudarmstadt.ukp.dkpro.semantics.type.CohesionIndicator"

PARAM_STRING_METHOD = "getStringRepresentation"

PARAM_UNIQUE_TERMS = true

EsaAnnotationPairSemRelAnnotator: Calculates semantic relatedness values for the edges that have been created in the previous step and adds according SemanticRelatedness annotations. This component uses explicit semantic analysis (ESA) such as described in Section 2.1.3, applied to the Wiktionary[7] knowledge repository. The configuration is as follows:

PARAM_INDEX = ""

PARAM_CACHED_INDEX = *Wiktionary index file*

PARAM_SCORE_CACHE_SIZE = 50

PARAM_VECTOR_CACHE_SIZE = 50

XmiWriterCasConsumer: Saves the whole document, including all annotations, to an XMI file. Configuration:

[7] http://en.wiktionary.org/

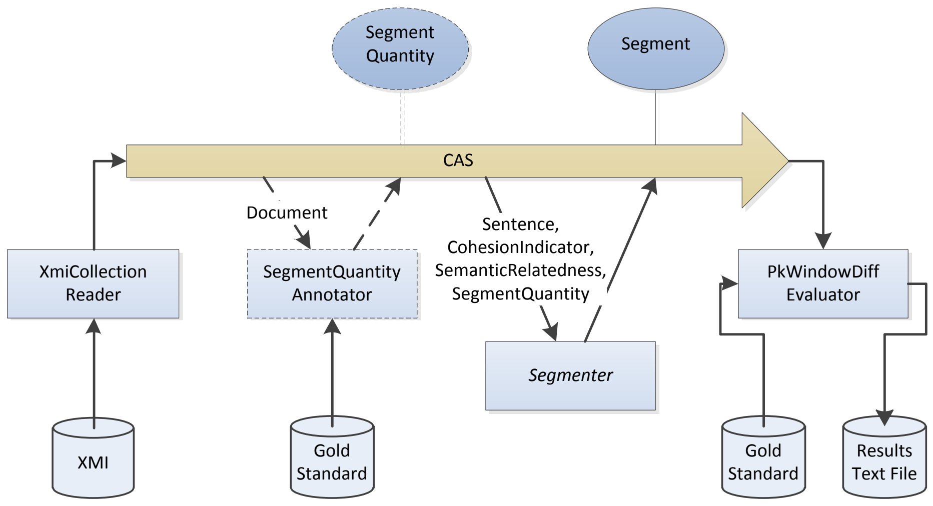


Figure 5.3: UIMA evaluation pipeline

PARAM_COMPRESS = true

5.4 Evaluation

The evaluation pipeline is shown in Figure 5.3. The design is the same as for the preprocessing pipeline. Additionally, dashed lines indicate optional elements of the pipeline.

The used components are shortly described here:

XmiCollectionReader: Is the counterpart of the XmiWriterCasConsumer and reads document contents and annotations from XMI files in a certain directory.

SegmentQuantityAnnotator*: Looks up the number of desired segments according to the gold standard (see Section 4.1) and adds a corresponding SegmentQuantity annotation to the document. This component is only applied if the number of segments should be provided to the segmentation algorithm. The following parameters must be set:

PARAM_GOLD_STANDARD_INPUT_DIRECTORY = *Directory with gold standard segment files*

PARAM_PATH_LEVELS_TO_CARRY_OVER_FROM_INPUT_TO_GOLD_STANDARD_FILE = *Integer value*
The meaning of this value is the same as explained with the AnnotationOffsetWriter in Section 5.2.

Segmenter: Executes a segmentation algorithm and adds appropriate Segment annotations. This component is to be understood abstract, i.e., in praxis, at this place a concrete segmentation component is applied. Concrete segmentation components are described in Sections 5.5 and 5.6.

PkWindowDiffEvaluator*: Compares the calculated segmentation such as given by the Segment annotations with the desired gold standard segmentation such as defined by the gold standard files. Calculates P_k and WindowDiff values for each document in the pipeline and collects them. After the pipeline has finished, averaged values are printed to an output file. An example output file is shown in Figure 5.4: Each line contains results of a certain group of analyzed documents. In this case, we used Choi's corpus which is divided into files with names containing one of the strings "3-5", "6-8", "9-11", and "3-11". Thus, we defined appropriate regular strings matchings those groups of files to obtain separate results for them. The last row contains results for all documents.

```
                         | Avg. (P_k) | Avg. (WindowDiff) | Avg. (# Segment) | Avg. (! Segments) | #
---------------------------------------------------------------------------------------------------------
URIs matching ".*3-5.*"  | 0.134665   | 0.140993          | 9.01000          | 10.0000           | 100
URIs matching ".*6-8.*"  | 0.116584   | 0.117495          | 9.12000          | 10.0000           | 100
URIs matching ".*9-11.*" | 0.106371   | 0.108329          | 9.14000          | 10.0000           | 100
URIs matching ".*3-11.*" | 0.148934   | 0.159004          | 8.42250          | 10.0000           | 400
All documents            | 0.136194   | 0.143262          | 8.70857          | 10.0000           | 700
```

Figure 5.4: Example output file of the PkWindowDiffEvaluator

The second and third columns contain P$_k$ and WindowDiff results; the fourth column presents the average number of calculated segments, the fifth column shows the average number of desired segments, and the last column simply informs about the number of documents belonging to the current document group.
The parameters of the component are defined as following:

PARAM_ANNOTATION_TYPE = "de.tudarmstadt.ukp.dkpro.semantics.type.Segment"

PARAM_GOLD_STANDARD_INPUT_DIRECTORY = *Directory with gold standard segment files*

PARAM_PATH_LEVELS_TO_CARRY_OVER_FROM_INPUT_TO_GOLD_STANDARD_FILE = *Integer value*
This value is defined as for the AnnotationOffsetWriter in Section 5.2.

PARAM_MALIOUTOV_EVALUATION = false
If this is true, P$_k$ and WindowDiff are not only calculated with our implementation, but also with an implementation of Malioutov he used for evaluation of his Minimum Cut Model [MB06]. However, evaluation with Malioutov's implementation is much slower than with ours.

PARAM_OUTPUT_FILE_NAME = *Path to results file*

PARAM_EVALUATION_LEVEL_TYPE = "de.tudarmstadt.ukp.dkpro.core.type.Token"
P$_k$ and WindowDiff can be calculated both on token and sentence level (see Section 4.2.3). This parameter constitutes the level on which to evaluate segmentations.

PARAM_URI_REGEX_GROUPS = *Array of regular expression strings*
These regular expressions denote groups of files which should be evaluated separately. A processed file is only included into a group if its URI matches the regular expression of the group. E.g., if the value of this parameter is {".*txt", ".*ref", ".*dat"}, the output file will contain separate result rows for documents which have a filename ending with "txt", "ref", and "dat".

PARAM_PK = true
Constitutes whether to use the P$_k$ measure.

PARAM_WINDOW_DIFF = true
Constitutes whether to use the WindowDiff measure.

PARAM_K_VALUES = { 0.5 }
This array of float values constitutes which values for k should be used for P$_k$ and WindowDiff. The values are relative to the average segment length, i.e., the value 0.5 indicates that k will be half the average segment length which is the standard configuration for those measures.

PARAM_SEGMENT_QUANTITY = true
Constitutes whether the number of calculated segments should be printed to the output file.

PARAM_GOLD_STANDARD_SEGMENT_QUANTITY = true
Constitutes whether the number of segments in the gold standard should be printed to the output file.

5.5 Cluster Blocks

The component

de.tudarmstadt.ukp.dkpro.semantics.segmentation.annotator.ClusterBlockSegmenter

implements the Cluster Blocks algorithm described in Section 3.1. The parameters have the following values in our pipelines[8]:

PARAM_ANNOTATION_TYPE = "de.tudarmstadt.ukp.dkpro.semantics.type.CohesionIndicator"
: This parameter denotes the type of annotations to be considered for clustering: In the semantic graph clustering is applied to, features of this type constitute the vertices, even if they are not connected by SemanticRelatedness annotations. (This is for the case that SemanticRelatedness annotations with weak relatedness values have been removed earlier since the previously connected terms should not be ignored in this case.)

PARAM_STRING_METHOD = "getStringRepresentation"
: This method of the annotations of the type defined with the previously described parameter is used to obtain the term string of the annotations to be used for the semantic graph. This might differ from the original token text, e.g., if lemmatization has been applied.

PARAM_CLUSTERER = "de.tudarmstadt.ukp.dkpro.semantics.segmentation.graph.GirvanNewmanClusterer"
: This parameter defines the clusterer with which to perform the clustering on the semantic graph. This must be the name of a class implementing the interface

de.tudarmstadt.ukp.dkpro.semantics.segmentation.graph.Clusterer.

The GirvanNewmanClusterer is our implementation of Girvan's and Newman's clustering method [NG04]. We have based our implementation on the EdgeBetweenness component of the JUNG framework[9]. We have extended this clustering implementation by automatic finding of the optimum clustering according to the modularity values of clusterings (see Section 3.1.4).

PARAM_GIRVAN_NEWMAN_CLUSTERER_SMOOTH_MASK: If the parameter PARAM_CLUSTERER is not defined and, thus, the GirvanNewmanClusterer is used, this parameter sets the mask which is used for smoothing the modularity values of calculated clusterings. This influences the selection of the optimum clustering. For details, see step 6 in Section 3.1.4.

PARAM_MAX_DELIMITING_BLOCK_SENTENCES: Defines the maximum number of sentences within a block which do not contain any word of the associated cluster.

PARAM_MIN_BLOCK_LENGTH: Defines the minimum number of sentences a block must consist of.

PARAM_MIN_SEGMENT_LENGTH: This float value constitutes the minimum length of a segment, relative to the average segment length. E.g., if the hypothesized number of segments is 10, the document contains 100 sentences, and this parameter is 0.5, each segment must contain at least $0.5 \cdot \frac{100}{10} = 5$ sentences.

PARAM_BOUNDARY_STRENGTH_SMOOTH_MASK: Defines a Float array which is applied as a smoothing mask to the list of boundary strength values. E.g., the smoothing suggested in Section 3.1.6 can be achieved with the mask $(1, 2, 1)$.

Additionally, before executing the actual segmenter component, a sparsificator is executed which removes edges from the semantic graph with low relatedness values. This component is implemented in

[8] Values which are not given here are provided in the Evaluation chapter.
[9] Java Universal Network/Graph framework; http://jung.sourceforge.net/

de.tudarmstadt.ukp.dkpro.semantics.segmentation.annotator.SemanticRelatednessSparsificator

and has the following parameters:

PARAM_MODE: This parameter must be one of the constants MODE_THRESHOLD and MODE_REMOVE_RATIO. The threshold mode removes all edges with a relatedness value lower than a constant. The ratio mode removes a certain ratio of edges where those with lower relatedness values are removed preferably.

PARAM_VALUE: Defines the threshold and ratio value, respectively. It must be a float value between 0 and 1.

PARAM_MAX_RELATIONS: This integer value defines the maximum number of SemanticRelatedness annotations to retain. If there are more annotations after application of the threshold or the ratio sparsification, excess annotations with the lowest relatedness values are removed.

PARAM_RESPECTED_ANNOTATION_TYPE = "de.tudarmstadt.ukp.dkpro.semantics.type.CohesionIndicator"
Defines the type of annotations which are candidates for being connected in the semantic graph. This influences the number of removed edges in the ratio mode since the totally possible number of edges depends on the annotation type.

5.6 Maximum Compact Segments

The Compact Segments algorithm is implemented in the component class

de.tudarmstadt.ukp.dkpro.semantics.segmentation.annotator.MaximumCompactnessSegmenter.

It has the following configuration parameters and parameter values in our pipeline:

PARAM_ANNOTATION_TYPE = "de.tudarmstadt.ukp.dkpro.semantics.type.CohesionIndicator"
As for the ClusterBlockSegmenter, this parameter defines the type of annotations which are part of semantic graphs.

PARAM_STRING_METHOD = "getStringRepresentation"
This, again, denotes the name of the method of annotations which returns the string representation of terms.

PARAM_MAX_SEGMENT_LENGTH = 1
This float value defines the maximum length of a segment to be considered during analyzation of a document. The length is relative to the document length. I.e., with the value 1, every possible segment length is considered.

PARAM_MAX_UNIT_COUNT = 100
As explained in Section 3.2.2, blocks are considered as smallest units during analyzation in this algorithm. This parameter denotes the maximum number of blocks. If more sentences than blocks are existing, blocks will contain multiple sentences. If this parameter is 0, every sentence will constitute a separate block. The rationale of this block limitation is to avoid runtime complexity of compactness and boundary strength value calculation to become cubic in the number of terms.

PARAM_ALPHA = 0.5
The α value in equation (3.1).

PARAM_BETA = 1
The β value in equation (3.1).

PARAM_GAMMA = 0.6
: The γ value in equation (3.1).

PARAM_D = 0.05
: The d value in equation (3.1).

5.7 Other Issues of Implementation

Besides the components described in the previous sections we have also implemented, and adapted respectively, some more components which are presented in this section.

For evaluation purposes, we applied implementations of other segmentation algorithms, namely Freddy Choi's C99 and Marti Hearst's TextTiling (see Section 2.3). Those implementations were already available as UIMA components and, hence, could uniformly be inserted into the evaluation pipeline (Figure 5.3) as "Segmenter" components. Unfortunately, existing implementations did not provide a possibility to pre-define the number of segments to be calculated which would be desirable for comparability. Therefore, we adapted the existing C99 UIMA implementation

de.tudarmstadt.ukp.dkpro.semantics.annotator.segmentation.C99segmenter

so that it made use of SegmentQuantity annotations if existing. The TextTiling implementation was more difficult to adapt; thus, we contented ourselves with C99 and left adaptation of TextTiling open for future work.

Furthermore, we implemented a degenerated segmenter in the class

de.tudarmstadt.ukp.dkpro.semantics.segmentation.annotator.DegeneratedSegmenter.

This component is a segmenter which places segment boundaries either at every sentence boundary, at no sentence boundary, or in equal distances. The component is configured with the following parameters:

PARAM_MODE: a Mode constant
: Mode constants are defined in the same component and comprise the following values: NO_SEGMENTS (only one segment for the whole document), ALL_SEGMENTS (segment boundary at each boundary of the considered annotation type, e.g., at each sentence boundary), FIXED_NUMBER_OF_SEGMENTS (as many equally sized segments as defined with the PARAM_COUNT_SEGMENTS parameter or, if available, with existing SegmentQuantity annotations).

PARAM_SEGMENTATION_LEVEL_TYPE: the class name of the annotation type which to separate into segments
: As we have chosen sentences to be the smallest unit of segmentation, we used the class name

  de.tudarmstadt.ukp.dkpro.core.type.Sentence.

PARAM_COUNT_SEGMENTS: the number of equally sized segments to create in mode FIXED_NUMBER_OF_SEGMENTS
: This parameter is not used if a SegmentQuantity annotation is available which provides the desired number of segments.

5.8 Component Overview

This section provides an overview of all UIMA components which have been implemented in the course of the thesis. Table 5.1 contains the following columns:

Type: The type of the component. This is one of CollectionReader, Annotator, and Consumer.

Name: The name of the component. The package of all listed components is

de.tudarmstadt.ukp.dkpro.semantics.segmentation.X

where *X* is, according to the type, one of "reader", "annotator", and "consumer". Abstract components are rendered in italic font.

Extends: The superclass of the component. All properties of the superclass also count for this component. Particularly, the values of the columns *Needs* and *Edits/Adds/Removes* should be minded.

Needs: The annotation types the component needs. The most types are standard DKPro types. New types are located in the package

de.tudarmstadt.ukp.dkpro.semantics.type.

Types of annotations which can be used by a component but are not necessary are given in square brackets.

Edits/Adds/Removes: Annotations of the types listed here are edited in some way. The DocumentMetaData annotation is created by all CollectionReaders, but not mentioned in the table.

Description: A short description of the purpose of the component.

Type	**Name**	**Extends**	**Needs**	**Edits/Adds/Removes**	**Description**
CollectionReader	ChoiDataSetReader	FileSystemReader	–	Sentence, Segment	Reads text files in Choi's format (see Figure 4.5).
CollectionReader	WikipediaAllArticlesReader	WikipediaSegmentationReader	–	–	Reads all articles except for disambiguation and redirection pages.
CollectionReader	WikipediaCategoryReader	WikipediaSegmentationReader	–	–	Reads all articles of a certain Wikipedia category.
CollectionReader	WikipediaIdFileReader	WikipediaIdReader	–	–	Reads all articles with IDs which occur in a text file listing IDs row by row.
CollectionReader	*WikipediaIdReader*	WikipediaSegmentationReader	–	–	Abstract component which reads all articles with certain IDs.
CollectionReader	*WikipediaSegmentationReader*	–	–	Segment	Connects to a Wikipedia database and uses JWPL in order to receive parsed articles. Adds Segment annotations based on a configurable granularity of the section level.
CollectionReader	WikipediaTemplateUsingArticlesReader	WikipediaSegmentationReader	–	–	Reads all articles which use a certain Wikipedia template. (Slow! Iterates over all articles.)
Annotator	AnnotationRemover	–	–	–	Removes annotations of configurable types.
Annotator	ClusterBlockSegmenter	–	Sentence, SemanticRelatedness, [SegmentQuantity]	Segment	Implementation of the Cluster Blocks algorithm (see Section 5.5)
Annotator	CohesionIndicatorAnnotator	–	Lemma	CohesionIndicator	Wraps Lemma annotations in CohesionIndicator annotations.
Annotator	DegeneratedSegmenter	–	[SegmentQuantity]	Segment	Implementation of baseline segmenter algorithms (see Section 5.7)
Annotator	GoldStandardSegmenter	–	DocumentMetaData	Segment	Adds Segment annotations according to a given gold standard.
Annotator	MaximumCompactnessSegmenter	–	Sentence, SemanticRelatedness, [SegmentQuantity]	Segment	Implementation of the Compact Segments algorithm (see Section 5.6)
Annotator	SegmentQuantityAnnotator	–	DocumentMetaData	SegmentQuantity	Adds a SegmentQuantity annotation which indicates the desired number of segments according to a gold standard.
Annotator	SemanticRelatednessSparsificator	–	SemanticRelatedness	SemanticRelatedness	Removes edges with weak relatedness values from the semantic graph.
Consumer	AnnotationOffsetWriter	–	[DocumentMetaData]	–	Writes character boundary indexes of annotations of a configurable type line-by-line to a file.
Consumer	ChoiFormatWriter	–	Sentence, Segment, [Document-MetaData]	–	Writes the CAS content in Choi's format (see Figure 4.5) to a text file.
Consumer	CorpusStatisticsEvaluator	DefaultSegmentationEvaluator	Token, Sentence	–	Collects token, sentence, and segment statistics of all processed documents.
Consumer	*DefaultSegmentationEvaluator*	SegmentationEvaluator	–	–	Provides methods for collecting results to subclasses and saves collected results to a file.
Consumer	PkWindowDiffEvaluator	DefaultSegmentationEvaluator	Segment	–	Collects P_k and WindowDiff results for the processed documents.
Consumer	PlainTextWriter	–	[DocumentMetaData]	–	Writes the pure text of the document to a file.
Consumer	*SegmentationEvaluator*	–	DocumentMetaData	–	Provides gold standard segments to subclasses.

Table 5.1: Overview of implemented UIMA components

6 Summary

This thesis pursued three intentions: To give an overview of existing text segmentation algorithms, to examine the possibility of exploiting semantic graphs in two new approaches, and to evaluate the new approaches based on the metrics P_k and WindowDiff and, among others, with a new Wikipedia-based corpus.

Before going into details of existing approaches, the thesis gave an overview of the main concepts of Natural Language Processing which one should be aware of when concerning oneself with text segmentation approaches. Possible categorizations of text segmentation algorithms have been suggested. In presentation of existing algorithms, we concentrated on works which exploit lexical cohesion and are linear and unsupervised. Lexical cohesion can manifest itself in word reiterations as well as in other features pertaining to the lexical value of words. We found works on text segmentation which based on word reiterations only, such as Hearst's TextTiling [Hea93] or also Choi's C99 [Cho00], but many algorithms also take into account other features. This can, for instance, be categories of words in dictionaries such as applied by Okumura and Honda [OH94], or co-occurent frequency vectors such as used by Ferret [Fer07]. We have seen that there are at least four types that can be distinguished with respect to the way in which algorithms find segment boundaries: 1. *Lexical Scores* are calculated in some works which express cohesion or also semantic dissimilarity between blocks of texts. According to the scores which are assigned to possible segments or possible segment boundaries, the final segmentation is chosen. 2. *Lexical Chains* are applied in order to find lexically coherent strings in a text. Based on the found chains, segment boundaries are chosen. 3. *Clustering* methods are applied, whether to matrices [Cho00] or to graphs [MB06], in order to find an optimum segmentation. 4. *Probabilistic Models* are implanted such as done by Utiyama and Isahara [UI01], finding an optimum segmentation with respect to certain probability definitions of word co-occurrences, segment lengths, or cue words.

Applications of text segmentation have been presented: For instance, text summarization can be enhanced by exploiting calculated segments in order to compose reasonable summaries covering all segments. In information retrieval systems, text segmentation can improve user contentment by presenting only relevant passages, and, at least as important as that, it can enhance recall and precision by using smaller pieces of texts for building the index. Further applications have been identified in the fields of language modeling, hyptertext linking, and anaphora resolution.

Two new approaches of text segmentations have been introduced. They focused on exploitation of semantic graphs which have in recent work not extensively been examined for their usefulness in text segmentation although many applications to other fields such as text summarization, keyphrase extraction, and query answering can be found. The first algorithm, *Cluster Blocks*, aims at finding topically related groups of vocabulary by performing a clustering on the semantic graph. These clusters are then assigned to blocks of sentences in the text, similarly to lexical chains. Segment boundaries are chosen at position with a high number of beginning or ending blocks. We have justified this procedure by giving an example of a text where one could obviously draw reasonable boundaries according to the distribution of topically related words in the text. Because of the similarity to lexical chain approaches in final finding segment boundaries, one could, consequently, classify Cluster Blocks into the group "Lexical Chains" we have defined in the Related Work section. The "Clustering" group is less appropriate because the clustering which is performed in our algorithm is not the final step which finds the segments; it is merely a preperation step which yields topical clusters of the text.

Compact Segments, the second new algorithm, finds an optimum segmentation with respect to segment quality values which are assigned to every segment candidate. The quality value depends on the two criteria compactness expressing the inner cohesion of a segment candidate and boundary strength which indicates the lexical dissimilarity of the candidate and adjacent sentences. Both values are calculated with respect to the semantic graph. We motivated this with an example that showed how compactness value for a given semantic subgraph of a segment candidate indicates the semantic cohesion. A dynamic programming method has been presented for optimization although other methods are possible, too, such as solving a shortest-path problem of a graph with appropriate edge weights. As this algorithm primarily bases on the segment quality values which can be understood as a kind of complex values for lexical cohesion, the group "Lexical Scores" of our classification scheme would best meet the algorithm's characteristics.

The thesis then described the methodology which has been applied in order to evaluate the quality of the new algorithms compared to baseline and other algorithms. Particularly, P_k and WindowDiff metrics have been presented and advantages of WindowDiff have been outlined. Both metrics are error metrics yielding values between 0 and 1 (0 and 100%, respectively), i.e., values close to 0 indicate a better quality. The metrics compare calculated segmentations with gold standard segmentations and penalize wrong boundary positions, however, handling near misses leniently which makes them more appropriate for evaluating text segmentation algorithms than, for instance, recall and precision, well known metrics stemming from evaluation of information retrieval systems. P_k and WindowDiff can be applied both on token and on sentence base; our decision for the first variant has been justified with an example. During evaluation, we have noted that P_k and WindowDiff might not really be appropriate for evaluation of segmentations on natural documents: The one "correct segmentation" according to which those metrics are calculated can hardly be defined for such documents.

Four corpora have been used in evaluation one of which, based on Wikipedia, we have generated for this thesis. Characteristics of the corpora have been summarized, and we have pointed out that they must be divided into two groups: two artificial corpora whose documents consist of concatenations of different document parts, and two natural corpora with single-topic documents which are divided into sub topics.

The process of tuning involved different parameters for our two algorithms. For Cluster Blocks, it could be seen that usage of semantic relations yields better results than pertaining to pure word reiterations for building blocks. For Compact Segments, we achieved best results for equally weighted compactness and boundary strength values.

During evaluation on other corpora, for the Cluster Blocks algorithm, detecting the number of actual segments turned out to be a hard problem since the number of clusters does not, as we originally believed, exhibit a dependency on the number of segments. When the number of segments was provided to the algorithm, it achieved results about 32% for P_k and WindowDiff on Choi's corpus which is significantly worse than Choi's C99 but better than baseline algorithms and still better than Hearst's TextTiling. Compact Segments was on the same quality level as C99 (about 12% P_k and 13% WindowDiff on Choi's corpus), but could not keep up with Utiyama's and Isahara's U00 and LCseg of Galley et al. which achieve values of 10.5% for P_k and 11.5% WindowDiff on the same corpus. For large numbers of segments, Compact Segments exhibited slight quality problems.

Concerning the different types of corpora, results revealed significant differences between natural and artificial corpora as natural documents are much more difficult to segment correctly according to the only correct gold standard.

The previous chapter has finally documented the UIMA pipelines which have been used for corpus generation and evaluation of the new algorithms. Newly implemented UIMA components have been summed up in an overview table.

6.1 Future Work

In evaluation, we have seen that the Cluster Blocks algorithm indeed achieves better results if not only word repetitions are used for building blocks, but also semantical clusters. However, the clusters obtained from Girvan's and Newton's clustering algorithm have not always satisfied the expectations: Often, many very small clusters of one or two terms were found, and one or two very big clusters contained nearly half the vocabulary of the text. This suggests that results of the Cluster Blocks algorithms could be enhanced significantly if the clusters were more appropriate and conformed better to the topics of the text. For that purpose, one could examine clustering results with different semantic measures than we used. The semantic relatedness values we obtained from the Wikitionary index (see Section 5.3) are often very low and differ only insignificantly. Better results may be achieved using the Wikipedia index, for instance. Also, another clustering algorithm would be worth a try although Grineva et al. [GGL09] claim to have yielded good results with Girvan's algorithm for finding topical clusters. As a more complicated idea, one could also consider a clustering algorithm which takes into account edge weights: In our present algorithm, we discard edges with low semantic relatedness values according to a threshold or a quota, and then perform the clustering with the remaining edges, ignoring edge weights. If edge weights would be respected by the clustering algorithm, one could, instead, apply it to the complete graph where edges with weight 0 would be considered as "not important". This would also promise more reasonable clusters; however, we did not find an appropriate clustering method for weighted graphs so far and, thus, fell back on Girvan's well-proven algorithm for unweighted graphs.

Cluster Blocks can also be improved with respect to automatic detection of the number of boundaries. This is in the current version, as described in Section 3.1.6, done by using a cutoff value for the boundary strengths, based on mean value and standard deviation. However, this did not prove to be very stable as for some texts the algorithm assumed more than 30 boundaries where only 10 have been in the gold standard. We found Hearst's TextTiling to run into similar problems during estimating the number of segment boundaries on the same way; so far, however, we did not find a better solution.

For the Compact Segments algorithm, a main point of improvement could be, in our opinion, the generation of even more expressive compactness and boundary strength values. This might partially be achieved by using other semantic resources since, as mentioned, the Wiktionary index often does not deliver that strong semantic relatedness values.

One could also think about taking into account the frequency of words which, in semantic graphs, originally vanishes because every word is represented exactly once. For instance, an alternative graph representation could be used in which the number of vertices for a word matches the number of its occurrences, and edge weights between vertices for equal words might be set to 1. This would, consequently, lead to higher compactness values for texts containing many word reiterations.

Tuning of the number of segments is currently done with the parameter d (see Section 4.4.2) which tends to be quite small for reasonable numbers of segmentations which impairs reliability of this method since little variation may have big impact on the number of estimated segments. Therefore, another, more reliable way for this estimation would be desirable.

Besides functionality, runtime is an issue which is definitely worth improving it: Both algorithms need the complete semantic graph to work properly which is costly to build, particularly due to the high number of semantic relatedness values to be calculated for different pairs of terms. This preprocessing took – per document – up to an hour of calculation time on a 2 GHz machine. In order to enhance this, instant calculation of semantic relatedness values must possibly replaced by a large database containing relatedness values for most relevant pairs of terms of the corpus. Additionally, more features, e.g., adjectives and verbs, could be left out in order to obtain a smaller semantic graph.

Runtime of the Cluster Blocks algorithm itself which has, in the current version, complexity $O\left(T^5\right)$ (see Section 3.1.7) might be enhanced by a less complex clustering algorithm which allows for a faster selection of the optimum clustering: As Girvan's and Newman's clustering algorithm does not naturally deliver an optimum clustering but many possible clusterings, each of these have to be analyzed for their quality which is a main reason for the high complexity of the algorithm. Nevertheless, while preprocessing with building the semantic graph takes dozens of minutes, the actual algorithm could be executed mostly in less than a minute.

The Compact Segments algorithm had in our experiments with a complexity of $O\left(T^3\right)$ an acceptable runtime of about 10 seconds per medium-length[1] document (ignoring preprocessing) which is mainly brought about by optimization with dynamic programming (see Section 3.2.6). Hence, this complexity cannot be reduced significantly.

[1] ca. 2000 tokens

List of Figures

List of Tables

Bibliography

[BBL97] Doug Beeferman, Adam Berger, and John Lafferty. Text segmentation using exponential models. In *In Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 35–46, 1997.

[BBL99] Doug Beeferman, Adam Berger, and John Lafferty. Statistical models for text segmentation. In *Machine Learning*, pages 177–210, 1999.

[BE97] Regina Barzilay and Michael Elhadad. Using lexical chains for text summarization. In *In Proceedings of the ACL Workshop on Intelligent Scalable Text Summarization*, pages 10–17, 1997.

[BRS92] Rodrigo A. Botafogo, Ehud Rivlin, and Ben Shneiderman. Structural analysis of hypertexts: identifying hierarchies and useful metrics. *ACM Trans. Inf. Syst.*, 10(2):142–180, 1992.

[Cho99] F. Y. Y. Choi. Jtexttile: A free platform independent text segmentation algorithm, 1999.

[Cho00] Freddy Y. Y. Choi. Advances in domain independent linear text segmentation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 26–33, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[EB08] Jacob Eisenstein and Regina Barzilay. Bayesian unsupervised topic segmentation. In *EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 334–343, Morristown, NJ, USA, 2008. Association for Computational Linguistics.

[Eis09] Jacob Eisenstein. Hierarchical text segmentation from multi-scale lexical cohesion. In *NAACL '09: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 353–361, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

[ER03] Leo Egghe and Ronald Rousseau. A measure for the cohesion of weighted networks. *J. Am. Soc. Inf. Sci. Technol.*, 54(3):193–202, 2003.

[ESK01] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding topics in collections of documents: A shared nearest neighbor approach. In *In Proceedings of Text Mine'01, First SIAM International Conference on Data Mining*, 2001.

[Fel98] Christiane Fellbaum. *WordNet: An electronic lexical database*. MIT Press, 1998.

[Fer07] Olivier Ferret. Finding document topics for improving topic segmentation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 480–487, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[GGL09] Maria Grineva, Maxim Grinev, and Dmitry Lizorkin. Extracting key terms from noisy and multitheme documents. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 661–670, New York, NY, USA, 2009. ACM.

[GM07] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1606–1611, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[GMFLJ03] Michel Galley, Kathleen McKeown, Eric Fosler-Lussier, and Hongyan Jing. Discourse segmentation of multi-party conversation. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 562–569, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[Hea93] Marti A. Hearst. Texttiling: A quantitative approach to discourse segmentation, 1993.

[Hea94] Marti A. Hearst. Multi-paragraph segmentation of expository text, 1994.

[Hea97] Marti A. Hearst. Texttiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64, 1997.

[HH76] M. A. K. Halliday and R. Hasan. *Cohesion in English*. Longman, London, 1976.

[HP93] Marti A. Hearst and Christian Plaunt. Subtopic structuring for full-length document access. In *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 59–68, New York, NY, USA, 1993. ACM.

[KF67] H. Kucera and W. N. Francis. *Computational analysis of present-day American English*. Brown University Press, Providence, RI, 1967.

[KF93] Hideki Kozima and Teiji Furugori. Similarity between words computed by spreading activation on an english dictionary. In *Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics*, pages 232–239, Morristown, NJ, USA, 1993. Association for Computational Linguistics.

[Koz93] Hideki Kozima. Text segmentation based on similarity between words. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 286–288, Morristown, NJ, USA, 1993. Association for Computational Linguistics.

[LALS07] Sylvain Lamprier, Tassadit Amghar, Bernard Levrat, and Frederic Saubion. On evaluation methodologies for text segmentation algorithms. In *ICTAI '07: Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, pages 19–26, Washington, DC, USA, 2007. IEEE Computer Society.

[Lin98] Dekang Lin. An information-theoretic definition of similarity. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 296–304, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[MB06] Igor Malioutov and Regina Barzilay. Minimum cut model for spoken lecture segmentation. In *In Proceedings of the Annual Meeting of the Association for Computational Linguistics (COLING-ACL 2006*, pages 25–32, 2006.

[MH91] J. Morris and G. Hirst. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational Linguistics*, 17:21–48, 1991.

[MH06] Saif Mohammad and Graeme Hirst. Distributional measures of concept-distance: a task-oriented evaluation. In *EMNLP '06: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 35–43, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[MH10] Meghana Marathe and Graeme Hirst. *Computational Linguistics and Intelligent Text Processing*, chapter Lexical Chains Using Distributional Measures of Concept Distance, pages 291–302. Springer Berlin / Heidelberg, 2010.

[MZM$^+$08] Christof Müller, Torsten Zesch, Mark-Christoph Müller, Delphine Bernhard, Kateryna Ignatova, Iryna Gurevych, and Max Mühlhäuser. Flexible uima components for information retrieval research. In *Proceedings of the LREC 2008 Workshop 'Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP'*, pages 24–27, Marrakech, Morocco, Mai 2008.

[NG04] M E Newman and M Girvan. Finding and evaluating community structure in networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 69(2):026113.1–15, February 2004.

[OH94] Manabu Okumura and Takeo Honda. Word sense disambiguation and text segmentation based on lexical cohesion, 1994.

[PC97] Jay M. Ponte and W. Bruce Croft. Text segmentation by topic. In *ECDL '97: Proceedings of the First European Conference on Research and Advanced Technology for Digital Libraries*, pages 113–125, London, UK, 1997. Springer-Verlag.

[PGKT06] Matthew Purver, Thomas L. Griffiths, Konrad P. Körding, and Joshua B. Tenenbaum. Unsupervised topic modelling for multi-party spoken discourse. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 17–24, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[PH02] Lev Pevzner and Marti A. Hearst. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28:1–19, 2002.

[Por97] M. F. Porter. *An algorithm for suffix stripping*, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[Rey98] Jeffrey C. Reynar. Topic segmentation: Algorithms and applications, 1998.

[SAB93] Gerard Salton, J. Allan, and C. Buckley. Approaches to passage retrieval in full text information systems, 1993.

[SM86] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[SM02] H. Grogory Silber and Kathleen F. McCoy. Efficiently computed lexical chains as an intermediate representation for automatic text summarization. *Comput. Linguist.*, 28(4):487–496, 2002.

[TVN10] George Tsatsaronis, Iraklis Varlamis, and Kjetil Norvag. Semanticrank: Ranking keywords and sentences using semantic graphs. In *Proceedings of COLING'2010, Beijing, China, August 2010*, 2010.

[UI01] Masao Utiyama and Hitoshi Isahara. A statistical model for domain-independent text segmentation. In *ACL '01: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 499–506, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[XC96] Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 4–11, New York, NY, USA, 1996. ACM.

[Yaa97] Yaakov Yaari. Segmentation of expository texts by hierarchical agglomerative clustering. In *Proceedings of RANLP'97*, Bulgaria, 1997.

[Zes09] Torsten Zesch. *Study of Semantic Relatedness of Words Using Collaboratively Constructed Semantic Resources*. PhD thesis, Darmstadt University of Technology, 2009.

[ZMG08] Torsten Zesch, Christof Müller, and Iryna Gurevych. Extracting lexical semantic knowledge from wikipedia and wiktionary, 2008.