Learning to Segment Web-Pages Using Repeat Analysis

Stefanos Angelidis

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2013

Abstract

In this thesis we investigate the novel approach of applying repeat analysis techniques for the task of web-page segmentation. Web-page segmentation is considered in the context of Information Retrieval's content extraction. Previous web-page segmentation approaches have mainly focused on heuristically analysing the visual appearance of web-pages in order to identify its main segments. Our approach is novel, in the sense that it utilizes principled, machine learning ideas, borrowed from the repeat analysis literature, in order to approach this task. For this purpose a multi-phase system consisting of phases to sequentially represent and analyse the web-page's source code was built. In order to evaluate the system's performance, we designed and performed a number of experiments, each focusing on different hypothesis, regarding the usefulness of each of the system's component. The obtained results indicated that our approach is very promising and lends itself to further investigation of this novel idea.

Acknowledgements

First and foremost, I would like to thank Prof. Victor Lavrenko for the outstanding level of supervision he provided me with. This thesis could not have been completed without his constant and valuable guidance. I would also like to thank my family, my friends and especially anyone who actively supported me throughout this project. This thesis could not have been completed without you.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Stefanos Angelidis*)

To my family.
To my old friend, Zibo.

Table of Contents

1 Introduction **1**

1.1 Problem Definition 3

1.2 Project Aim and Objectives 4

1.3 Dissertation Structure 5

2 Related Work **7**

2.1 Automatic Sequence Segmentation 8

2.2 Web-Page Segmentation 9

2.2.1 Single Content Section Extraction 10

2.2.2 General Case: Multiple Sections Segmentation 12

2.2.3 Critical Assessment of Existing Segmentation Approaches . . 13

2.3 Repeat-related Sequence Mining 15

2.3.1 Dispersed, Variable Length Repeats 15

2.3.2 Dot-Plots 16

2.3.3 Suffix Trees, Suffix Arrays and Enhanced Suffix Arrays . . . 18

2.4 Evaluating Segmentation Tasks 19

3 Methodology **21**

3.1 Preliminaries 21

3.1.1 Sequence Terminology and Notation 22

3.1.2 Sequential Representation of Web-Pages 24

3.1.3 Defining Interspersed Segmentation 26

3.2 System Overview 27

3.3 Components Description 28

3.3.1 Sequence Translation 28

3.3.2 Repeat Extraction 31

3.3.3 Repeat Density Analysis 38

3.3.4 Segments Extraction 42

4 Experiments **49**

4.1 Dataset 50

4.2 Evaluation Methodology 52

4.2.1 Ground Truth Generation 53

4.2.2 Evaluation Metrics 54

4.3 Experiments and Results 57

4.3.1 Preliminaries 58

4.3.2 Evaluating the Repeat Extraction Component 60

4.3.3 Evaluating the Effect of Sequence Translation 65

4.3.4 Effect of *minlen* and *minrep* parameters 68

4.3.5 Effect of Clustering on Segment Extraction 70

5 Conclusions **73**

5.1 Future Work 75

Bibliography **77**

A Configuration Sweeps **81**

List of Figures

1.1 Overview of a search engine's indexing process 2
1.2 Examples of single and multiple content web-pages 4

2.1 Sequence segmentation output from different domains 8
2.2 Examples of single content web-pages 10
2.3 Tag plateau algorithm visualized 11
2.4 Sample segmentation of VIPS algorithm 12
2.5 Taxonomy of the major repeat-related problems 15
2.6 Example of exact and approximate, dispersed repeats 16
2.7 Examples of Synthesized Dot-Plots 17
2.8 Suffix Trees/Arrays applications and the traversal types they require . 18

3.1 Examples of sub-sequences, maximal and supermaximal repeats . . . 23
3.2 Example of discourse and interspersed segmentations 26
3.3 Full system overview illustration 27
3.4 Character-based sequence representation 29
3.5 Simple token-based sequence representation 30
3.6 Extended token-based sequence representation 30
3.7 High-level design of the repeat extraction phase 31
3.8 Example of dot-plot visualization from a toy page sequence 33
3.9 Simplified example of repeat identification using suffix and prefix information 34
3.10 Example output of the dotplot component 35
3.11 Length-thresholded repeat extraction using enhanced suffix arrays . . 36
3.12 Length- and frequency-thresholded repeat extraction using enhanced suffix arrays 37
3.13 Simple example of kernel smoothing 39
3.14 Repeat density analysis of repeats extracted from the dot-plot component 40

3.15 Repeat density analysis of repeats extracted from the enhanced suffix array component 41
3.16 Peak extraction applied to a repeat density function 42
3.17 Example segmentation output with naive extraction 44
3.18 Illustration of peak clustering before and after cluster selection 46

4.1 Examples of web-pages from our Dataset 52
4.2 Generalization performance per source (optimized on all sources) . . 61
4.3 $\ddot{k}$-κ value of 7 in-domain experiments 64
4.4 Generalization performance of translation policies per source 67
4.5 Segmentation performance effect of *minlen* parameter in character- and token-based sequences 69

List of Tables

3.1 Total number of content characters in each cluster 47

4.1 Description of web-page sources used in the dataset 51
4.2 Dataset statistics 51
4.3 Summary of system phases and their alternatives/parameters 58
4.4 Stripped-down en. suffix array generalization performance (trained on all sources) 60
4.5 k-precision and k-recall values 62
4.6 T-test significance testing for the basic system configuration 62
4.7 Separately optimized parameters for each source 63
4.8 Mean $\ddot{k}$-κ value, averaged over the 7 system runs (one per source) . . 63
4.9 k-precision and k-recall values 64
4.10 Optimized parameters for each sequence translation policy 66
4.11 Generalization performance of each translation policy 66
4.12 T-test significance testing for both translation policies 68
4.13 Generalization performance comparison with and without segment clustering 71
4.14 T-tests between clustering configurations and our baseline 71
4.15 T-tests between clustering configurations and their naive alternatives . 72

Chapter 1

Introduction

The world wide web is, by many means, the moving force of modern life. It is being used by hundreds of millions users every day for professional, commercial, informational or entertainment reasons. Its vast size, its exponential growth over the years and the constantly updating content available in it, makes the web the most impressive information data source to ever exist, far exceeding any physical collections of data. In addition, the ease of use that characterizes it, makes it accessible to people from all over the world and can be the mean of distant communication, that would never be possible otherwise.

The abundance of information available in the web could, in no way, be accessible to its users without the existence of the *Information Retrieval* (IR) research area and, particularly, search engines. Search engines have been a crucial part of the web since the very early stages it became widely accessible. They provide a very easy way for any user to identify desired content, while, essentially, filtering the remaining – and, mostly, irrelevant – information. In fact, at any given moment a user's information needs only cover an extremely small portion of the available content. This makes search engines both undeniably useful for the efficient use of the world wide web, as well as impressively accurate when it comes to providing precisely what is requested by the user, as fast as possible.

Such a level of accuracy and response time would never be possible without extremely sophisticated underlying algorithmic and storage structure. Search engines are highly complex systems, that consist of many different components working independently and in cooperation, in order to provide the best possible experience to the user.

All search engines, both commercially available ones – like Google, Bing, etc.–, as well as specific purpose ones – engines for legal, medical, biological etc. professional use – depend hugely on the existence of large amounts of data, from which their IR systems will be created. Especially in the case of commercial, web-related search engines, where the spectrum of covered content is the broader, the process of acquiring, storing, and utilizing the available data is more prominent than ever.

The general process that relates to the aforementioned gathering and proper utilization of data is commonly referred to as *Search Engine Indexing*. Search Engine Indexing is a multi-phase procedure, which can be simplistically described as having 4 major stages of operation. These 4 stages are illustrated in Fig. 1.1.

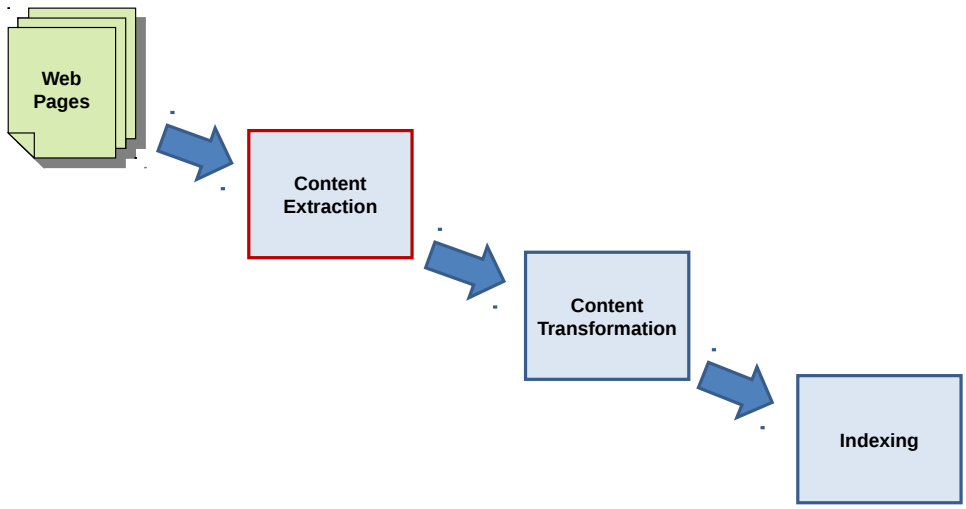


Figure 1.1: *Overview of a search engine's indexing process*

In simple terms the operational *pipeline* works as follows. Initially, as much data available – in the form of web-pages – as possible must be gathered and be constantly inspected for potential updates, through a procedure known as *web-crawling*. Once the raw data are made available, every web-page must go through the phase of content extraction, i.e. the process of identifying and extracting the useful information in them, by distinguishing the parts of the web-page that are the most important content-wise. Subsequently, the extracted content is *transformed* into a suitable representation, so that it can be processed by the various IR algorithms that may be in use in the particular system. Finally, these representations are *indexed* so that they can be stored and, subsequently, retrieved in a very efficient way.

This thesis deals with the phase of *content extraction*. Not all information available in web-pages is of equal importance, either to the user who visits it or to the IR system that needs to index it. Web-pages are filled with a plethora of elements; some exist for navigational or visual cohesion reasons, others for advertising or copyright purposes, and, obviously, some correspond to the actual content that is intended to be the center-piece of the page. A search engine, whose indexing component is not capable of distinguishing between the content section of the page and any other surrounding elements, will probably have disadvantages in terms of performance accuracy. Considering every section of the page equally important can mislead the algorithms that utilize the indexed web-pages. While identifying these web-page section is an easy task for a human, it is not a trivially automated procedure.

As a consequence of that, accurate techniques, capable of automatically identifying the regions of content significance, have to be developed in order to aid the proper indexing of web-pages. This thesis deals with a particular type of content extraction problem, which is defined in the following section.

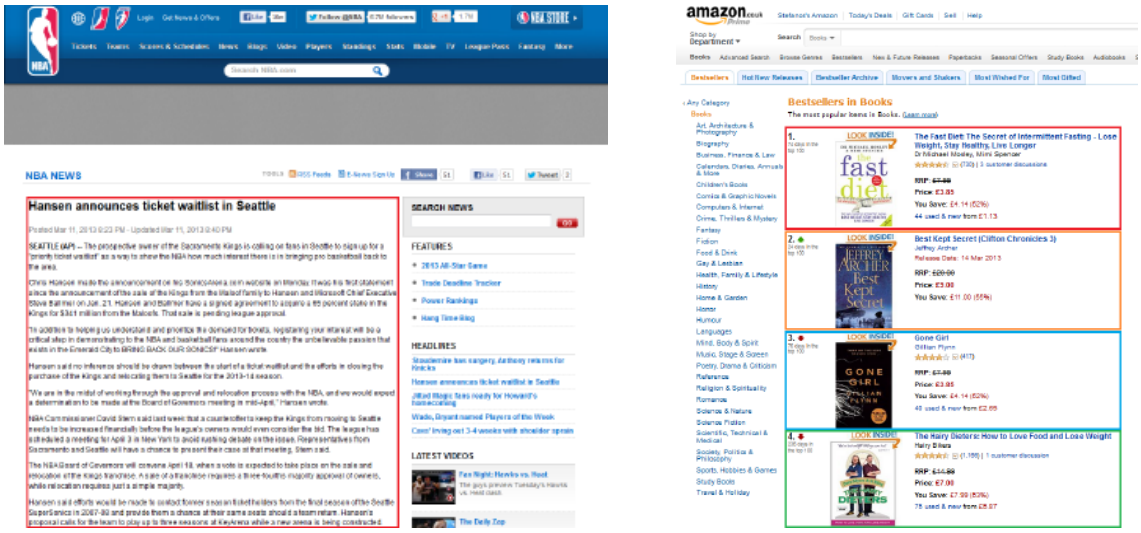
1.1 Problem Definition

The task of web-page content extraction can be divided into two families of sub-tasks depending on the type of web-page that is to be analysed. The first refers to the task of extracting content information from web-pages that only contain a single content section. Typically, such web-pages are news stories, articles, pages that contain static information about organizations or people etc. An example is shown in Fig. 1.2(a). This type of content extraction will not be the focus of this thesis.

For this project, we will deal with the task of content extraction for multiple content section web-pages. Pages containing multiple content sections are found in abundance in the web. Online store product lists, search engine results, video lists, user communities etc. are only a few examples. An online store's product list is shown in Fig. 1.2(b).

The task of extracting these sections is not as trivial as its single-content counterpart. Ideally, given a web-page containing multiple content sections, structured in a visually similar way, a content extraction component of an IR system should be able to identify the parts of the web-page where the content sections lie. Additionally, it

should be able to extract each section as an independent logical unit, part of a group of similar elements. This type of content extraction task can be viewed as a web-page segmentation problem.



(a) (b)

Figure 1.2: *Examples of (a) single and (b) multiple content web-pages*

Our approach is based on a sequential view of web-pages. We expect that, due to the heavily structured nature of html source code and the visual similarity of the content sections, significant repetitive patterns should be evident in the sequential representation of web-pages. By applying techniques and ideas from the repeat analysis literature – a novel approach for the web-page segmentation task –, we hope to identify underlying patterns, in order to extract logical units/sub-sequences that correspond to the content sections of the page.

1.2 Project Aim and Objectives

Our aim is to build a system, based on machine learning principles, that will utilize repeat analysis techniques as a mean of identifying content sections of multi-content web-pages. Due to the novelty of this approach, our main goal is to assess the suitability of repeat analysis for such a task.

In order to accomplish this goal, a number of main objectives were set for this project. These are:

1. Create an appropriate benchmark dataset to be used for training and evaluating systems, for the task of segmenting multi-content web-pages. This includes creating a ground truth for the task.

2. Study the sequence- and repeat- analysis literature in order to identify tools and techniques that could potentially be suitable for this problem.

3. Build a novel web-page segmentation system by adapting and combining these tools in order to fit the requirements of such a task.

4. Identify appropriate evaluation metrics to be used as an objective function, for training and evaluating the performance of the system.

5. Assess the quality of segmentation by thoroughly and critically evaluating the system.

1.3 Dissertation Structure

This thesis is structure in such a way, that the reader should be able to comprehend the ideas behind this project and the course of work that was put into it.

In chapter 2, we present an extensive review of the related literature that this project relates to and was influenced by. The reader will be introduced to the research fields of sequence segmentation and repeat analysis. In chapter 3 a detailed, top-down description of the system that was built will be given. A formal definition of the necessary terminology will be followed by a high-level system overview. Then a step by step presentation of the system's components will be given. Chapter 4 includes any information related to the system's evaluation. A description of the benchmark dataset will be followed by definitions of evaluation metrics and a thorough set of evaluation results. Finally, chapter 5 will include a summary of the main conclusions that were drawn in the process of this project, and any ideas for further extensions and variations that our work could lead to.

Chapter 2

Related Work

The problem that this dissertation deals with is closely related to a number of research fields. This happens both in terms of where it draws its motivation from, as well as regarding previous research work, which it either directly borrows ideas from or is heavily influenced by.

The task of extracting multiple content segments from web-pages has not been addressed extensively enough in the context of *Information Retrieval* (IR) and, in particular, its indexing process. However, a fair amount of research work regarding the task of *Web-Page Segmentation* has been done for purposes not related to IR. Additionally, *Content Extraction* for web-wages containing a single content section is a task that has been addressed and for which well-defined solutions have been proposed and applied with success as we will see later in this chapter.

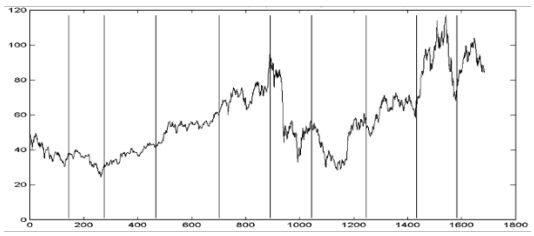
The core idea of this project has its roots in a research area that is usually referred to as *Sequence Analysis* or *Sequence Mining*. The most common applications of Sequence Mining can be found in the field of bioinformatics, which has been the most active field regarding sequence-related research for the past few decades.

In this chapter, firstly a small review of various segmentation tasks will be provided in section 2.1. This aims to help the reader get a sense of the broader context of sequence segmentation and the variety of domains it is applicable. Secondly, the previous work that has been done in both IR-driven content extraction, as well as the more general task of web-page segmentation will be reviewed in section 2.2. Next, a set of methods and tools from the Sequence Analysis literature which are in some way related to this project will be presented (section 2.3). Finally, in section 2.4, a small summary of the most frequently used evaluation metrics for segmentation tasks will provided.

2.1 Automatic Sequence Segmentation

The abundance of sequentially represented data and the inter-disciplinary need for their in-depth analysis has led to on-going research efforts in order to effectively extract information from them. Such data range from discrete representations such as textual or biological sequences to real-valued time-series of both scientific and industrial interest (speech, stock-market prices, weather measurements, web-usage click-through information etc.). Although varying in type and complexity, all meaningful sequential data tend not to behave randomly, but instead appear to contain particular patterns either in terms of distinguishable repetitiveness (e.g. DNA sequences) or as shifting trends of behaviour (e.g. stock-market, weather measurements).

Automatic sequence segmentation is the task of identifying homogeneous subsequences in sequential data. The nature of the sequence as well as the definition of homogeneity can vastly differ from one domain to another. This has led to the development of a large variety of approaches for solving such tasks.



(a) Stock-market time-series [1]

Police in Lebanon said that two Red Cross workers abducted last week are being held by Palestinian guerrillas led by terrorist Abu Nidal. Meanwhile, thousands of students returned to classes as Lebanon's state schools, most private schools and the American University of Beirut reopened after being closed for six months.

The White House said that a cyst removed Friday from Bush's right middle finger wasn't cancerous.
A presidential spokesman said a routine pathological examination was performed on the half-inch growth following the 25-minute surgical procedure at Walter Reed Army Medical Hospital.

Singapore's Lee Kuan Yew said he would step down as prime minister by the end of next year, ending three decades of nearly one-man rule in the island nation.
The 66-year-old Lee, in an interview with the British Broadcasting Corp., said he would hand over power to his deputy, Goh Chok Tong.

(b) Multi-topic text [2]

Figure 2.1: *Sequence segmentation output from different domains*

Figure 2.1 illustrates two examples of sequence segmentation tasks using a sample output from each. In 2.1(a) the time-series of a stock-market's index is segmented by identifying specific patterns in its behaviour. The vertical lines correspond to segment boundaries. In 2.1(b) a completely different type of sequence segmentation is shown. A textual sequence – part of a larger news stream – containing adjacent non-related news stories from different topics is the input of the segmentation task. The goal here is to identify the positions in the stream – dashed lines – that correspond to topic boundaries. Ideally, each segment in-between two consecutive boundaries should be identified as a single, independent news story.

[1] Figure from lai Chung et al. [2004].
[2] Figure from Ponte and Croft [1997].

It is evident that the research area of sequence segmentation covers a diverse family of problems. The field of Bioinformatics – since its emergence over two decades ago – has been actively leading research efforts in the area of genome sequence analysis in general and genome sequence segmentation in particular. The most prominent techniques are based on the statistical modelling of the segmentation task [Braun and Muller, 1998]. Another common task is the aforementioned topic segmentation. Similarly to genome sequence segmentation, the input is discrete valued, although in this case the definition of homogeneity is usually defined in terms of a language model or of sentence/paragraph similarity [Ponte and Croft, 1997]. An important example of a continuous version of such problems is speech segmentation, i.e. the task of segmenting audio signal of human speech into words or sentences [Shriberg et al., 2000].

Lastly, it should be noted that sequence segmentation may also act as an intermediate step towards solving other tasks. Lavrenko et al. [2000] proposed a novel idea for predicting stock prices trends; Stock market time-series were segmented in order to identify trends that were subsequently aligned to concurrent financial news stories. Features that were most indicative of upcoming price trends were then used to predict stock prices given unseen financial articles.

In the next section the focus will shift towards the related literature regarding the type of segmentation task that concerns this dissertation, i.e. web-page segmentation.

2.2 Web-Page Segmentation

As with most segmentation problems, the need for web-page segmentation has its roots in a variety of computer science fields and applications. With the emergence of the WWW, it quickly became obvious that effective content extraction techniques would be useful in order to gather and classify the huge amounts of information available by distinguishing between important and secondary parts of web-page. Later on, the establishment of internet access through screen constrained mobile devises (e.g. PDAs. smartphones, etc.) introduced another utility of web-page segmentation; since displaying the whole web-page to the user is inconvenient, a way to identify the most important parts of it became a necessity.

In sections 2.2.1 and 2.2.2 a review of some prominent approaches to the problem is provided. In the former, we focus on a well-established method for web-page segmentation in the context of IR-inspired content extraction. In the latter, we briefly present a number of proposed techniques that approach the task in a more general context. Finally, in 2.2.3 the limitations of the previously mentioned methods are discussed.

2.2.1 Single Content Section Extraction

As described in the introductory chapter, content extraction is an essential part of every web-related IR system. In its simplest form it can be seen as a specific case of a web-page segmentation task: Given a web-page that consists of a *single* "useful" content section surrounded by any less informative elements (navigational links, menus, advertisements, etc.), the goal is to find 2 appropriate boundaries such that the content section will lie in-between them, excluding any other element.

Figure 2.2: *Examples of single content web-pages*

Figure 2.2 illustrates two examples of single content web-pages. In both cases the page consists of a big block of text (i.e. the news story) as well as other, less relevant to the actual content, web-page elements. Distinguishing the main body of the article (marked in red in Fig. 2.2) and using it as the base for any IR- or classification-related

tasks will provide less misleading information about the page leading to more accurate results.

Probably the most well-established method for extracting a single content element from web-pages like the ones shown before is the one proposed by Finn et al. [2001]. Their simple, yet effective approach is based on viewing each web-page's source code as a sequence consisting of two kinds of tokens: HTML tag tokens and non-HTML tokens (i.e. words, numbers etc.). It makes intuitive sense that the part of the source that corresponds to the main content section will be mainly consisting of words and very few (if any) HTML tags. This is more obvious if we plot the number of HTML tags against the overall number of tokens that appear in such web-pages.

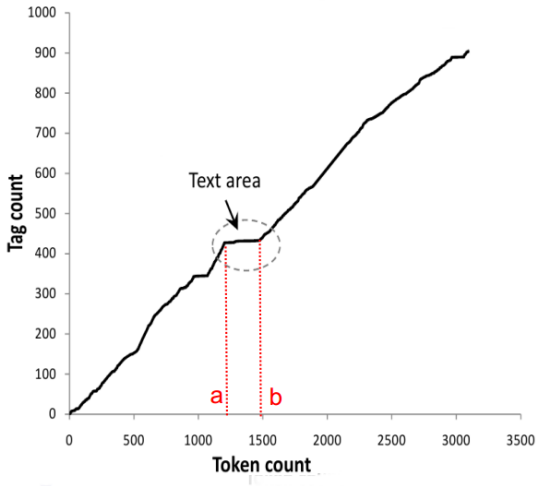


Figure 2.3: *Tag plateau algorithm visualized*[3]

The low density of html tags in the main content section of the page results in a flattened area in the plot, also known as the *tag plateau*. the point *a* and *b* mark the boundaries of the tag plateau which should, ideally, correspond to the start and the end of the content section.

Although far different from this dissertation's proposed method, the tag plateau algorithm adopts the sequential view of web-pages which is a key-element of this project as well. However, the fact that it is limited to only dealing with a single content section makes it unsuitable for the task of extracting multiple content sections that we are dealing with.

[3] Addison Wesley, 2008 ©.

2.2.2 General Case: Multiple Sections Segmentation

When not focusing on just a single content section as described in the previous section, web-page segmentation takes the more general form of finding a set of boundaries such that the resulting segments in-between consecutive boundaries are homogeneous in terms of either visual structure or content (usually the former is indicative of the latter). The motivation behind this kind of web-page segmentation differs from approach to approach, although the most prominent ones are duplicate detection for web-mining purposes, information extraction and identification of important page segments for displaying in screen-space constrained devices.

The most popular family of web-page segmentation approaches focuses on the visual interpretation of web-pages in order to identify coherent blocks that clearly differ from the rest of the page. The VIPS algorithm (**VI**sual-based **P**age **S**egmentation) [Cai et al., 2003] is an example of such a system. Their approach was to use heuristic rules for analysing the Document Object Model tree (DOM-tree) structure of a web-page. DOM tree nodes represent different parts of the web-page, while the nested nature of the tree allows for groupings of neighbouring HTML elements. While traversing the DOM tree of a page in a top-down manner, the VIPS algorithm compares various visual cues between neighbouring blocks of the page to evaluate the coherence of each traversed sub-tree (e.g. colour difference, text font, special styling tags etc.). A decision is made at each traversal step about whether to keep dividing the current sub-tree or not based on its visual coherence. A segmentation example of the VIPS algorithms is shown in Fig. 2.4. A web-page consisting of a series of user posts is shown with the corresponding DOM tree structure appearing on its left and the segmentation output of the VIPS algorithm appearing on its right.

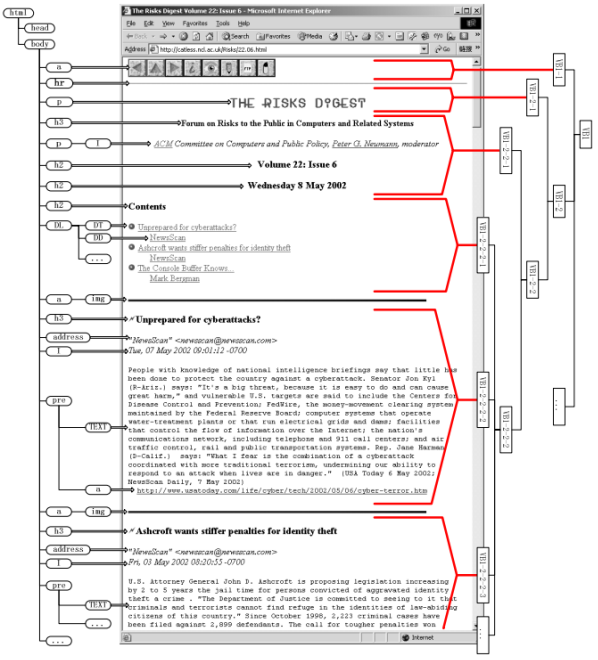


Figure 2.4: *Sample segmentation of VIPS algorithm*[4]

[4]Figure from Yu et al. [2003].

Another approach which borrows ideas from computer vision was proposed by Kohlschütter and Nejdl [2008]. In this case, the density of the textual parts of a web-page is analysed using vision techniques. Highly dense text portions of the page are more likely expected to be part of content sections of the page than not. The text density analysis results in a 1-dimensional representation of the web-page, where significant changes in the text density *slope* should be indicative of a boundary between neighbouring sections of the page. This technique is a departure from the heuristic approaches of earlier visual-based methods, as it focuses on modelling the segmentation task in a principled manner similarly to our approach.

Finally, another group of web-page segmentation approaches are based on graph-theoretic techniques. In such cases, a web-page is usually represented as a graph whose nodes are DOM tree elements and the edge weights between them represent how likely it is for these two nodes to be placed in the same segment or not [Chakrabarti et al., 2008]. The segmentation problem is then transformed into a graph-based optimization task where, given a training set of manually segmented web-pages, the goal is to learn the appropriate edge weights.

2.2.3 Critical Assessment of Existing Segmentation Approaches

The task of web-page segmentation has attracted significant research attention for a number of different reasons as shown in sections 2.2.1 and 2.2.2. Despite being a small sample of the related literature, the presented publications are indicative of the most dominant ideas and approaches on this task. Although often presenting promising results in the respective evaluation of their work, each approach has certain limitations. Moreover, the problem area of web-page segmentation lacks of concrete benchmark datasets and evaluation metrics for unbiased assessment and comparison of different techniques, which makes it hard to extract meaningful conclusions.

Finn's tag plateau algorithm for extracting the main text segment of single-content web-pages (2.2.1) is a well-established method, with satisfactory performance on the task it was developed for. Unfortunately, as previously discussed, it is only suited for single-content web-pages and it does not generalize well to the case of multi-content ones.

Multiple section segmentation methods are needed to overcome tag plateau's limitation. However, a number of drawbacks can also be identified for such approaches.

Heuristic-based techniques like the VIPS algorithm are – by nature – not flexible enough to be used in a wide scale. Coming up with heuristic rules is not a trivial task and there is no guarantee that they will result in similar performance when tested on unseen web-pages with different structure like the ones they were developed for. The always changing trends in the visual structure of web-pages in the WWW makes matters even worse for such techniques that cannot be dynamically re-trained. On the contrary, the aim of this dissertation's project was to develop a robust learning algorithm that would not rely on any heuristics and should be able to generalize to unseen web domains.

One major problem of all existing proposed web-page segmentation methods is the inconsistency regarding their experimental evaluation. Due to the lack of benchmark datasets specifically designed for this task, the common practice for each new effort is to train and test their method on newly created datasets.

Additionally – and even more importantly – there are no established and universally accepted evaluation metrics for this particular type of segmentation. Although a variety of metrics designed for segmentation problems do exist (see 2.4), none of them have been adapted or transformed for the particular case of web-pages. Instead, common evaluation techniques include survey-like evaluations using human judgement for assessing the quality of segmentation [Cai et al., 2003], implicit evaluation by using the segmentation output as an input for other applications [Yu et al., 2003] or borrowing evaluation metrics from non-segmentation tasks [Chakrabarti et al., 2008]. This project aims at both creating a diverse enough dataset for evaluating web-page segmentation methods as well as coming up with proper evaluation measures for such a task.

Finally, the IR-motivated nature of this project introduces another important factor. Since the goal is to extract sections of the web-page which correspond to the actual content information, it is crucial that the segmentation output includes only content sections and disregards any navigational elements, advertisements, headers/footers etc.. However, the vast majority of methods in related literature do not try to assess the content information of the output segments and only focus on segmenting the whole web-page into its structural blocks. Song et al. [2004] developed a method that, given a web-page already segmented into structural blocks, would evaluate the importance of each block in order to identify the ones containing the actual information content. Our project differs from any previous efforts, as it is designed to only extract those segments that correspond to content sections.

2.3 Repeat-related Sequence Mining

As described briefly in Chapter 1, the main idea of this project is to utilize repeat analysis as the basis of a web-page segmentation method. The term repeat analysis – also known as self-similarity analysis – refers to a particular family of sequence mining tasks that deal with analysing a single sequence in order to extract information about the presence (or absence) of repeated sub-sequences that occur in it.

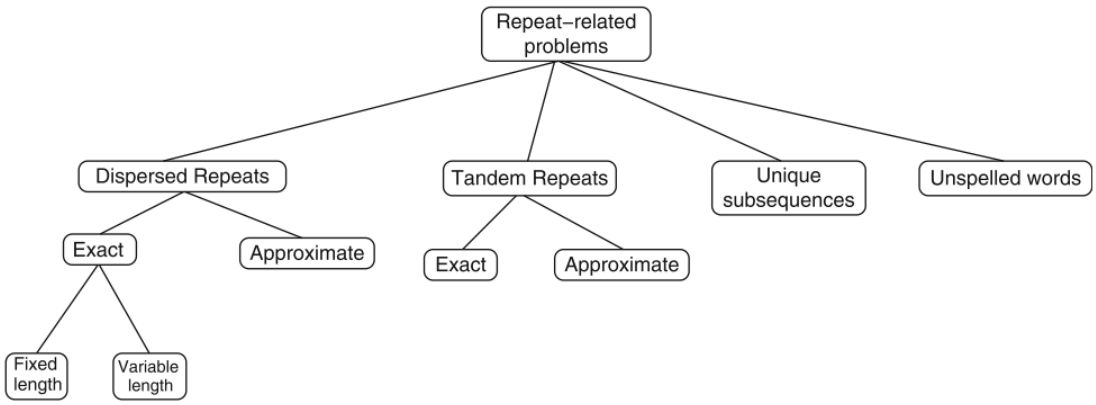


Figure 2.5: *Taxonomy of the major repeat-related problems*[5]

An overview of the most basic sequence-related problems associated with repeat analysis is provided in Figure 2.5. Tools and algorithms for efficiently extracting these types of sub-sequences have been and continue to be developed and used in a plethora of computer science fields. The main research efforts for the majority of these repeat-related problems have been – at least since the emergence of genomic sequence decoding – through the bio-informatics literature [Abouelhoda and Ghanem, 2010]. Additionally, repeat analysis has been frequently used in areas such as data compression – with the Lempel-Ziv-Welch algorithm being a notable example [Welch, 1984] –, frequent item set mining [Agrawal et al., 1994] etc.

In section 2.3.1 a short description of the subset of repeat-analysis problems that this project deals with is given. Then, in sections 2.3.2 and 2.3.3 the two repeat-related tools that are utilized in our proposed methods are presented.

2.3.1 Dispersed, Variable Length Repeats

For the scope of this project we will only focus on a subset of the aforementioned problem taxonomy, i.e. variable length, exact, dispersed repeats. Brief mentions of

[5]Figure taken from Abouelhoda and Ghanem [2010].

approximate repeats and how they could also be utilized will also appear. In this section we will provide some simple definitions on these types of repeats. More formal definitions related to sequences and repeats are given in Section 3.1.1.

A repeated pair is any pair of sub-sequences that occur in a single sequence that are either identical (exact repeat) or slightly altered by a few edit operations (approximate repeat). The dispersed property means that the two sub-sequences that form the repeated pair are not required to be adjacent. Finally, the variable length property indicates that the desired length of the repeat is not known beforehand. The most usual case of variable length repeats are maximal repeats, i.e. repeats that cannot be extended any further in either direction. Figure 2.6 shows both a maximal exact repeat (a), as well as an approximate one (b), where a character insertion (blue 'a') and a character replacement (red 't') has occurred.

(a) Maximal Exact Repeat (b) Approximate Repeat

Figure 2.6: *Example of exact and approximate, dispersed repeats*

2.3.2 Dot-Plots

Dot-plots are one of the oldest tools that have been used for sequence mining and for repeat-related problems in particular. Initially developed for genome sequence analysis [Gibbs and Mcintyre, 1970], dot-plots are a special type of recurrence plots. They can be used both for extracting intra-sequence repetitions as well as for identifying similarities between two sequences by plotting a sequence against itself or against a second sequence respectively.

Their construction is rather straightforward. By plotting a sequence against itself in its original order (in the case of intra-sequence analysis) we end up with a 2-dimensional, square grid where each cell corresponds to a pair of sequence positions (i, j). If sequence characters in positions i and j are identical then a *dot* is placed in the cell, otherwise it is left blank. This creates visual patterns that can be easily interpreted by humans or used to automatically extract self-similarity information.

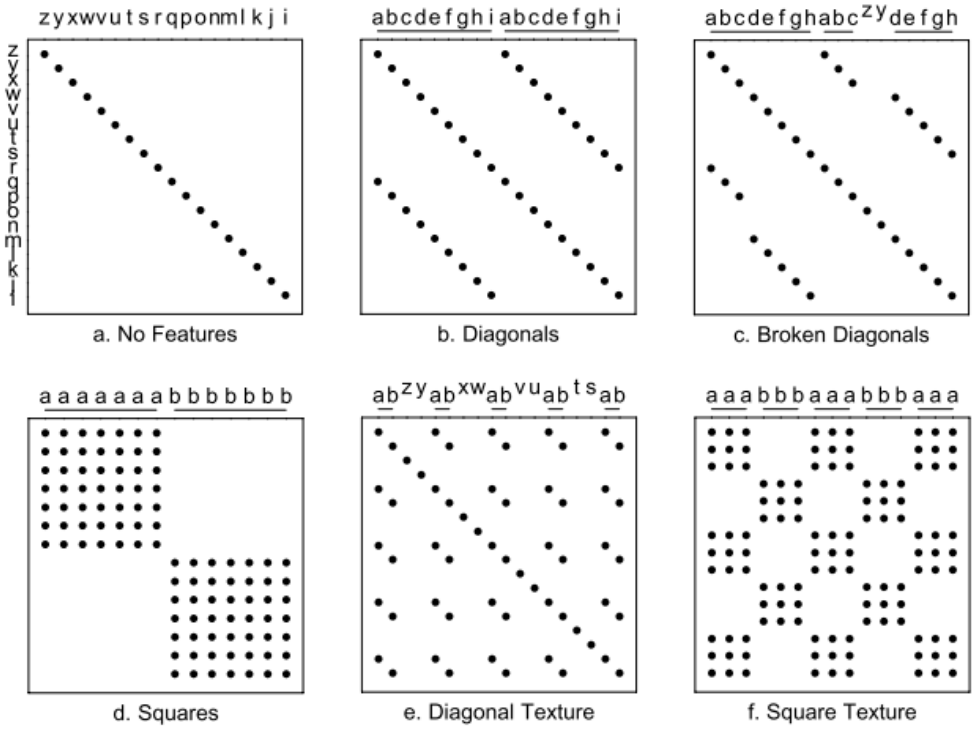


Figure 2.7: *Examples of synthesized Dot-Plots*[6]

Obviously the main diagonal of a self-similarity dot-plot is always an "unbroken" line since each character in position i matches itself. If no repetitions are evident in the sequence then the rest of the dot-plot is going to be blank (2.7 a). Exact repeated sub-strings appear as continuous lines off the main diagonal (2.7 b). If a repeat is approximate then the corresponding off-diagonal line is "broken" (2.7 c). Other interesting patterns can also appear: square regions in the case of single character repetitions (2.7 d, f), repetitions with intervening characters (2.7 e), etc. It is easily perceived that self-similarity dot-plots as the ones shown above are always going to be symmetric with respect to the main diagonal.

The same type of patterns will also be evident when analysing similarities between two sequences. However, it is obvious that in this case the dot-plot will neither be square (except if the sequences have equal lengths), nor symmetric and will not have an "unbroken" main diagonal. In the scope of our project, dot-plots will only be used for self-similarity analysis.

Dot-Plots have been used in genomic sequence analysis more often than in any other research field. Since their emergence [Gibbs and Mcintyre, 1970] they have been utilized for DNA, RNA and protein mining using increasingly sophisticated algo-

[6]Figure taken from Church and Helfman [1993].

rithms and representation variations [Brown et al., 1995; Dunham et al., 1999; Maizel and Lenk, 1981]. Althouth their main application lays in genetics, the context independent nature of dot-plots has made them useful in other sequence-related tasks as well. They allow the investigation of interesting patterns in any text or chronological event regardless of scientific domain. Such applications include, among others, software development redundancy identification [Church and Helfman, 1993] and version maintenance [Helfman, 1996], web-site evolutionary modification tracking [Bernstein et al., 1991], plagiarism detection [Helfman, 1994] etc.

2.3.3 Suffix Trees, Suffix Arrays and Enhanced Suffix Arrays

It could be argued that the most important and commonly used tool in string processing is the *suffix tree*. The suffix tree is a data structure that can be used to solve numerous sequence related problems with optimal time complexity in many cases [Gusfield, 1997].

Its importance is particularly obvious in the case of very large sequences whose analysis would otherwise be extremely time consuming. Due to this fact they have been extensively used to analyse whole genomes which in many cases can have lengths up to a few billion elements. Suffix trees are tree-like index structures that can be built in linear time with respect to the length of the sequence for which they are constructed and which, when traversed in the proper way, can provide efficient solutions to many repeat-related tasks. A summary of these tasks paired with the type of tree traversal they require is shown in Figure 2.8.

Application	Type of tree traversal		
	Bottom-up	Top-down	Suffix-links
Supermaximal repeats	√		
Maximal repeats	√		
Maximal repeated pairs	√		
Longest common substring	√		
All-pairs suffix-prefix matching	√		
Ziv–Lempel decomposition	√		
Common substrings of multiple strings	√	√	
Exact string matching		√	
Exact set matching		√	
Matching statistics		√	√
Construction of DAWGs		√	√

Figure 2.8: *Suffix Trees/Arrays applications and the traversal types they require*[7]

[7]Figure taken from Gusfield [1997].

Regardless of their prominent role in the sequence processing literature, suffix trees' use in actual implementations is very limited due to two main factors. Firstly, although they require $O(n)$ space complexity, their actual space consumption is at least 20 bytes per sequence element, which is unacceptable for genomic scale tasks [Kurtz, 1999]. Secondly, they suffer from poor memory locality which makes their performance poor despite their asymptotically linear time complexity [Abouelhoda et al., 2004; Delcher et al., 1999].

A simple and significantly more space efficient alternative to suffix trees is the *Suffix Array*. Introduced by Manber and Myers [1993], suffix arrays can be stored using only 4 bytes per sequence element and can be constructed in linear time [Kim et al., 2003]. However, the initial version of the suffix array could not achieve the optimal time efficiency of suffix trees for some repeat-related tasks. Almost one decade later, Abouelhoda et al. [2002] extended the suffix array data structure and introduced the *Enhanced Suffix Array*, which has since been proven able to solve any repeat-related task at least as efficiently as suffix trees do, by using only up to 6 bytes per sequence element [Abouelhoda et al., 2004].

More information on the specifications of enhanced suffix arrays, the algorithms that utilize them and how they were used in the scope of this project will be given in section 3.3.2.

2.4 Evaluating Segmentation Tasks

The unique nature of segmentation problems, as the ones discussed in the previous sections, makes the experimental evaluation of segmentation methods a non-trivial task. Even in the case where a ground-truth dataset denoting the reference segmentation does exist, it is hard to come up with a metric that properly accounts for how good a hypothesised segmentation is, while being tolerant enough to near misses.

Common evaluation metrics used throughout the machine learning research area are usually not applicable. Although many approaches have been proposed in various related domains, the most prominent and frequently used ones belong to a family of segmentation metrics that are referred to as *window-based metrics* [Beeferman et al., 1999; Niekrasz and Moore, 2010; Pevzner and Hearst, 2002].

The main idea behind this family of metrics is that, given a reference and a hypothesized segmentation of a sequence with N potential segmentation positions, a sliding window of length k is used which results in $N-k+1$ slices of the sequence. Then

the reference and hypothesized segmentations are compared either in terms of existence/absence [Beeferman et al., 1999] or in terms of the exact number of boundaries within each slice [Pevzner and Hearst, 2002]. The resulting number usually corresponds to the proportion of slices where the compared segmentations disagreed. These metrics have the advantage of introducing a near-miss tolerance factor in terms of the window length k.

More information regarding the evaluation metrics that will be used in this project are given in section 4.2.2.

Chapter 3

Methodology

This chapter provides a detailed description of the system that was built for this project. The methods that were designed and implemented in order to approach the problem of web-page segmentation from a content-extraction perspective will be given in a top-down manner. The solution will firstly be described in a high-level fashion by providing a full system overview and briefly commenting on the purpose of the components that constitute its operational phases. Subsequently, a detailed description of each phase will be provided while discussing the ideas that resulted in the particular design decision in favour of others. Some of the components will be clearly related to previous work described in chapter 2. This is because they are influenced by existing techniques or because they use tools off-the-shelf.

The outline of the chapter is the following: In Section 3.1 some necessary definitions and notation will be provided both for sequences in general as well as for web-pages as sequentially represented data. Section 3.2 contains a high-level description of the system and its main phases. In Section 3.3 the reader will be guided through the components from which each phase is built.

3.1 Preliminaries

It should be clear by now that most aspects of this project are closely related to sequence-analysis. In order to establish a common vocabulary that will be used throughout the rest of this thesis, this section's purpose is to formally define all necessary sequence terminology both in general terms (3.1.1) and specifically in relation to web-pages (3.1.2). Section 3.1.3 formally defines the type of sequence segmentation that this system deals with.

3.1.1 Sequence Terminology and Notation

The following definitions are commonly used in the sequence analysis literature. In particular, most of the terminology provided here is borrowed from the sequence mining survey of Abouelhoda and Ghanem [2010].

Sequences

Let Σ be an *ordered alphabet* of size $|\Sigma|$, consisting of a fixed number of unique, non-divisible *elements*[1]. We define S as a sequence of elements over Σ and its length $|S| = n$. We write $S[i]$ to denote the element of sequence S at position i, for $0 \leq i < n$ [2]. For $0 \leq i \leq j < n$, we define the *position pair* (i, j) that refers to the starting position i and ending position j of a sub-sequence of S. The actual content of this sub-sequence is denoted as $S[i..j]$. Finally, we define two special kinds of sub-sequences. The sub-sequence starting at the 0-th and ending at the i-th element of S is called a *prefix* of S and is denoted as $S[0..i]$. Similarly, the sub-sequence of S starting at position i and ending at the last element of S is called the i-th suffix of S and is denoted as $S[i..n-1]$, or using the more convenient notation $S(i)$.

Exact Repeats

Given a sequence S, a pair of positions $R = \langle (i_1, j_1), (i_2, j_2) \rangle$ is a *repeated pair* if and only if $(i_1, j_1) \neq (i_2, j_2)$ and $S[i_1..j_1] = S[i_2..j_2]$. The length of repeated pair R is $j_1 - i_1 + 1$. R is called *left maximal* if $S[i_1 - 1] \neq S[i_2 - 1]$ and *right maximal* if $S[j_1 + 1] \neq S[j_2 + 1]$. R is called *maximal* if it is both left and right maximal. In simpler terms, a repeated pair is called maximal if and only if it cannot be extended either to the left or to the right while still corresponding to identical sub-sequences. The sub-sequence ω is a (maximal) repeat if there exists a (maximal) repeated pair $\langle (i_1, j_1), (i_2, j_2) \rangle$ such that $\omega = S[i_1..j_1] = S[i_2..j_2]$[3]. Finally, a special type of maximal repeats are *supermaximal repeats*. A maximal repeat ω is called supermaximal if it never occurs as a sub-sequence of another maximal repeat.

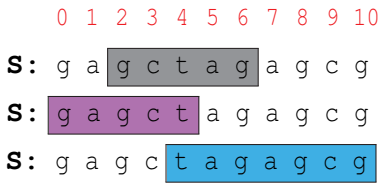
[1] In the context of sequences, an *element* that is part of an alphabet might not necessarily be of unit length. Not to be confused with the term *character* defined as a single text unit (alphanumerical, whitespace or punctuation symbol). Will be made clearer in section 3.1.2.

[2] The sequence indexing notation is zero-based, meaning that the starting element of sequence S is at position 0 and its last element is at position $n - 1$.

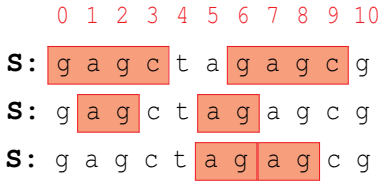
[3] In other words, the term *repeated pair* refers to the positions of the identical sub-sequences in S, whereas the term *repeat* refers to the actual content of the repeated sub-sequence.

Figure 3.1 illustrates the above definitions. $S =$ "gagctagagcg" is a sequence of length $n = 11$ which is defined over the alphabet $\Sigma = \{a, c, g, t\}$.

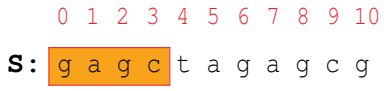
- Three sub-sequences are shown in 3.1(a): the sub-sequence $S[2..6]$ (gray), the prefix $S[0..4]$ (purple) and the suffix $S(4)$ (light blue).
- The three maximal repeated pairs of S (with $length > 1$) are illustrated in 3.1(b): $\langle(0,3),(6,9)\rangle$, $\langle(1,2),(5,6)\rangle$ and $\langle(5,6),(7,8)\rangle$. Note that repeated pair $\langle(1,2),(7,8)\rangle$ is **not** maximal since it can be extended in both directions.
- The only supermaximal repeated pair of S is $\langle(0,3),(6,9)\rangle$ shown in 3.1(c). Since the maximal repeat $\omega =$ 'ag' is a sub-sequence of maximal repeat $u =$ 'gagc', $\langle(1,2),(5,6)\rangle$ and $\langle(5,6),(7,8)\rangle$ are **not** supermaximal repeated pairs.



(a) A sub-sequence, a prefix and a suffix



(b) Maximal Repeats



(c) Supermaximal Repeat

Figure 3.1: *Examples of sub-sequences, maximal and supermaximal repeats of sequence $S =$ "gagctagagcg"*[4]

[4]Idea for figure borrowed from Abouelhoda and Ghanem [2010]

3.1.2 Sequential Representation of Web-Pages

Web-pages are, in reality, nothing more than simple text files containing mainly HTML source code. Their unique structure, however, allows them to be thought of and represented in more interesting ways. In particular it makes intuitive sense to think of web-pages both using their DOM-Tree representation, as well as using the more abstract view of them as a sequence of elements.

The DOM-Tree view of a web-page makes its illustration easier for a human to understand and allows for convenient processing of their nested structure in an automated manner. In the scope of this project, however, we will not consider the DOM-Tree representation of web-pages.

On the other hand, the sequential view of a web-page – in the way sequences were defined in section 3.1.1 – allows for increased level of flexibility when it comes to what is considered as a "*unique and non-divisible*" element[4]. In the same way that a plain textual object (e.g. a book) can be seen as a sequence of letters and symbols, as well as a sequence of words, sentences, paragraphs etc., the structure of web-pages allows for various levels of sequential abstraction based on the preferred *tokenization* method. In this context, tokenization is defined as the process of breaking up text into characters, words, phrases, or any other meaningful elements called tokens[5]. Different tokenization policies will result in different sequences derived from a single web-page.

In its most straightforward sequential form, a web-page W can be represented as a sequence of characters S_{char} over the character alphabet Σ_{char}. Σ_{char} consists of any unit-length alphanumeric, white-space or punctuation character found in the page's source code. From now on, this type of sequential representations will be referred to as *character-based sequences*.

Another possible tokenization policy is based on the fact that web-pages consist of two types of multi-character tokens: html tags and whitespace-separated, non-tag words[6] in-between of those tags. This tokenization policy results in a family of sequential representations, which will be referred to as *token-based sequences*. Depending on the alphabet used, different resulting sequences can be derived.

[4] See 3.1.1: Sequences

[5] http://en.wikipedia.org/wiki/Tokenization

[6] In this context, a "word" does not refer to the linguistic definition of a word but rather to any sequence of non-whitespace alphanumeric or symbol character.

The most basic token-based sequence type is the one already described in section 2.2.1: A sequence S_{bin} over the binary alphabet $\Sigma_{bin} = \{0,1\}$ where $S_{bin}[i] = 1$ if the i-th token of the web-page is an html tag and $S_{bin}[i] = 0$ otherwise.

For this project, however, we will only consider more sophisticated token-based sequence types. In particular, we will define two such representations: The first will be referred to as *simple token-based sequences* and the second as *extended token-based sequences*.

Given a web-page's source we define its *simple token-based sequence* representation S_{simple} over the alphabet Σ_{simple} as follows: Let T be the set of unique html tags found in the web-page without taking into account the various parameters that each tag may have. Additionally, let w_{span} be a special element denoting an unbroken span of consecutive whitespace-separated non-tag words. The alphabet of the simple token-based sequence S_{simple} is then defined as $\Sigma_{simple} = T \cup \{w_{span}\}$. Note that due to the definition of w_{span}, which can correspond to an arbitrary number of consecutive non-tag words, it is impossible to find two adjacent elements of S_{simple} that are both equal to w_{span}, i.e. if $S_{simple}[i] = w_{span}$ then $S_{simple}[i+1] \neq w_{span}$ for $0 \leq i < n-1$.

A variation of this representation is the *extended token-based sequence* S_{ext} over the alphabet Σ_{ext} that is defined as follows: Again, let T be the set of unique html tags found in the web-page without taking into account the various parameters that each tag may have. Additionally, let P be the set of html tag parameters found in the web-page without taking into account the actual values of each parameter. Finally, let w be a special element denoting a single non-tag word. The alphabet of the extended token-based sequence S_{ext} is then defined as $\Sigma_{ext} = T \cup P \cup \{w\}$. In contrast to the case of S_{simple}, in S_{ext} it is possible to have two or more adjacent elements all equal to w, since w only corresponds to a single non-tag word.

The process of constructing a sequential representation of any type from a web-page's source code will be referred to as *sequence translation*. The construction of character-based sequences is trivial since no actual processing takes place. Therefore, in the rest of the dissertation, character-based sequences will be thought of as having undergone *no* sequence translation. On the other hand, token-based sequences are the result of *simple* and *extended* sequence translation respectively.

Obviously, there are numerous other tokenization policies that could be considered. However, in the scope of this project only the 3 mentioned above will be used and assessed as they are expected to be representative enough for our purposes. More information regarding the translation process will be given in Section 3.3.1.

3.1.3 Defining Interspersed Segmentation

The most common way to formally define a sequence segmentation task is in terms of a *discourse* [Niekrasz and Moore, 2010]. Each sequence S of length n, which is to be segmented, is associated with an M-length array $\langle p_1, p_2, \ldots, p_M \rangle$ of *potential boundary positions*, which specify $M+1$ adjacent non-divisible elements $\langle [0, p_1), [p1, p_2), \ldots, [p_M, n] \rangle$.

A segmentation $\mathbf{X}$ is defined as a sequence of Boolean variables $\langle X_1, X_2, \ldots, X_M \rangle$, such that $X_m = 1$ if there is a boundary at p_m, and $X_m = 0$ otherwise.

However, in our task the segments to be extracted are not guaranteed to be adjacent – but may as well be –, since irrelevant web-page sections may or may not exist in-between two content segments. Additionally, they will definitely not cover the whole web-page sequence.

In order to address the unsuitability of the discourse segmentation definition, two types of boundaries are defined for our segmentation problem: opening and closing ones. In this case, a segment is defined as the sub-sequence in-between an opening and a closing boundary. We will now formally define this type of segmentation, which will be referred to as *interspersed* segmentation.

In the same way as before, let the sequence S of length n be associated with an M-length array $\langle p_1, p_2, \ldots, p_M \rangle$ of *potential boundary positions*, which specify $M+1$ adjacent non-divisible elements $\langle [0, p_1), [p1, p_2), \ldots, [p_M, n] \rangle$.

An *interspersed* segmentation $\ddot{\mathbf{X}}$ is defined as a sequence of paired Boolean variables $\langle \ddot{X}_1, \ddot{X}_2, \ldots, \ddot{X}_M \rangle$, such that:

$$\ddot{X}_m = \begin{cases} (1,0) & \text{if there is an opening boundary at } p_m \\ (0,1) & \text{if there is a closing boundary at } p_m \\ (1,1) & \text{if there is an opening } \textbf{and} \text{ a closing boundary at } p_m \\ (0,0) & \text{if there is } \textbf{no} \text{ boundary at } p_m \end{cases}$$

An illustration is provided in Fig. 3.2:

(a) (b)

Figure 3.2: *Examples of (a) discourse and (b) interspersed segmentations*

3.2 System Overview

The main aim of this project, as it has been described in the introductory chapter of this thesis, is to utilize the sequential nature of web-pages in order to analyse the presence of repeated patterns that will hopefully correspond to segments of the web-page exhibiting structural similarities while containing significant content.

By carefully reading the above description, one can identify the 4 main phases that should constitute the backbone of the system's design. Given the initial input – a single web-page – these 4 processing steps should work as a pipeline and produce a final output of the predicted web-page segments that were extracted. In general terms, these steps are:

1. **Sequence Translation:** Transform the raw source of the input web-page into a sequential representation.
2. **Repeat Extraction:** Given the constructed sequence, extract repeated sub-sequences that are considered significant enough.
3. **Repeat Density Analysis:** Analyse the extracted repeats in order to identify regions exhibiting high repetitiveness.
4. **Segments Extraction:** Transform these regions into actual segments that form the systems segmentation prediction.

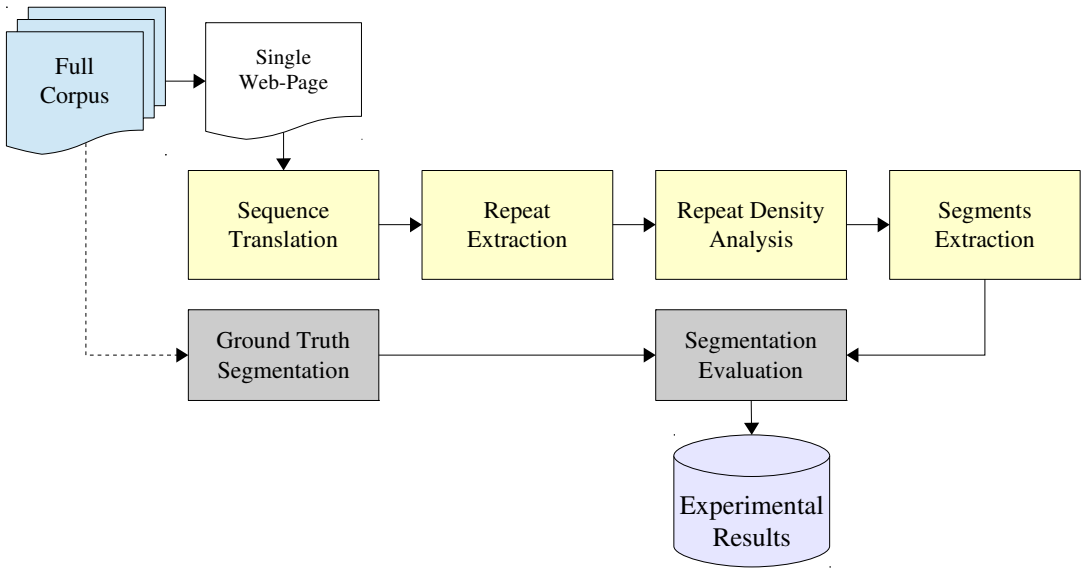


Figure 3.3: *Full system overview illustrating both processing (yellow) and evaluation (gray) components*

Figure 3.3 provides an overview of the full system design. The system's input is a single web-page which is part of a large corpus of web-pages. The main system pipeline is illustrated by the 4 basic system components shown in yellow colour. Each of these 4 components will be described in detail in Section 3.3. The components illustrated in gray colour correspond to the evaluation phase of the system that will be discussed in Chapter 4.

3.3 Components Description

In this section a step-by-step description of the 4 main components constituting our system is provided. Sub-sections 3.3.1-3.3.4 are intended to provide details on the design decisions for each component, the reasons that resulted in these decisions, major variations and alternative approaches (if any) that were implemented for each phase, as well as the most important parameters that are expected to affect the overall system performance.

3.3.1 Sequence Translation

The sequential representation of web-pages was discussed in detail in section 3.1.2 where the formal definitions and descriptions of 3 types of sequence representations were provided. Summarizing, the representation types considered in this project are:

- **Character-based sequences** *(no translation)*: Each character of the web-page source is translated into a single sequence element.
- **Simple token-based sequences** *(simple translation)*: Each html tag of the web-page source is translated into a single sequence element. Tag parameters are disregarded. Consecutive content words in-between tags are translated into a single element.
- **Extended token-based sequences** *(extended translation)*: Each html tag or tag parameter of the web-page is translated into a single sequence element. Values of tag parameters are disregarded. Each content word in between tags is translated into a single element.

The *sequence translation component* is responsible for transforming the raw input (page's source) into a sequence of elements suitable for the repeat analysis steps that follow. The translation process is rather straightforward. In the case of character-based sequence representation, the input web-page source remains intact and will subsequently be treated as a simple sequence of characters. In the case of the 2 token-based sequence representations the input web-page source is tokenized appropriately using regular expressions. The resulting sequences are defined over a newly created alphabet. Additionally, for each element, a pair of starting and ending positions that correspond to its boundaries in the non-translated, character-based sequence are also outputted.

Figures 3.4-3.6 illustrate the process for all 3 representations. A "toy" web-page source (black text) is used to show how each translation method produces a different sequence representation. The resulting sequences are coloured in such a way that each element has a different colour from its neighbours.

In Figure 3.4 the web-page undergoes no translation. A sequence S_{char}, with $|S_{char}| = 66$ (equal to the source's length) and a corresponding alphabet $\Sigma_{char} = \{c : \forall\, c \in S_{char}\}$ is constructed.

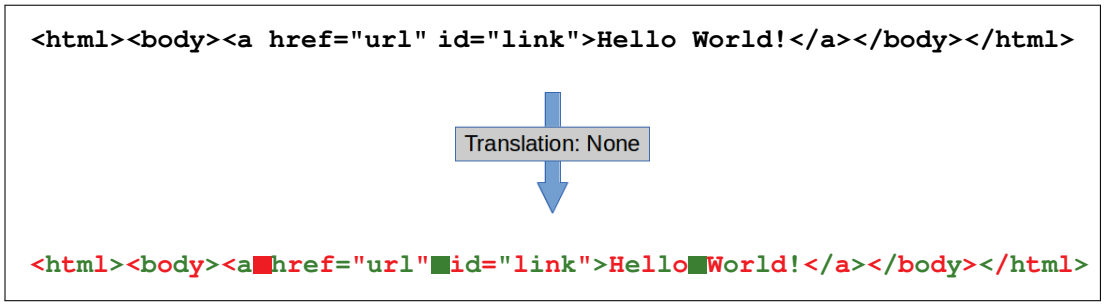


Figure 3.4: *Character-based sequence representation*

In Figure 3.5 the web-page undergoes simple token-based translation and is transformed into a sequence S_{simple}, with $|S_{simple}| = 7$ and a corresponding alphabet $\Sigma_{simple} = \{$<html>, <body>, <a>, , </body>, </html>, $w_{span}\}$. The starting and ending *position boundaries* that each element of S_{simple} maps to in the character-based sequence are also shown.

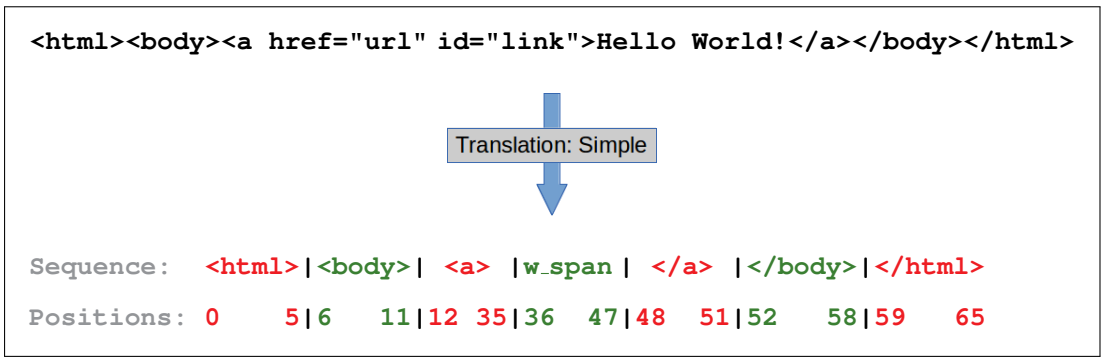


Figure 3.5: *Simple token-based sequence representation*

In Figure 3.6 the web-page undergoes extended token-based translation and is transformed into a sequence S_{ext}, with $|S_{ext}| = 10$ and a corresponding alphabet $\Sigma_{ext} = \{$<html>, <body>, <a>, , </body>, </html>, href, id, $w\}$. The starting and ending *position boundaries* that each element of S_{ext} maps to in the character-based sequence are also shown.

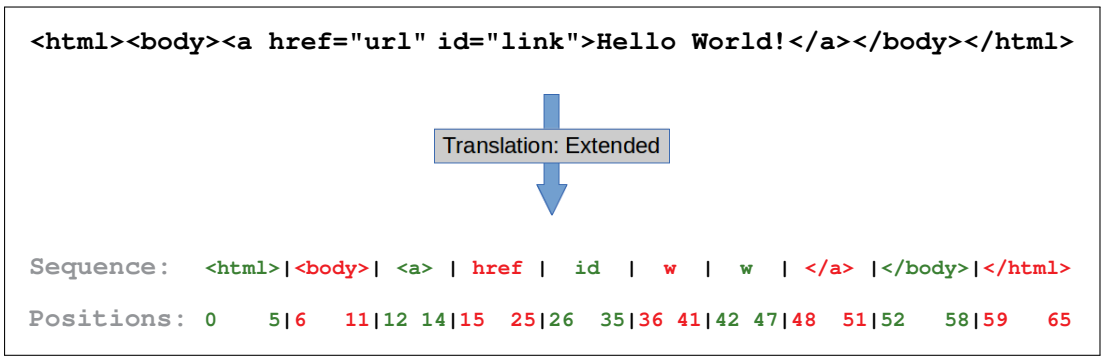
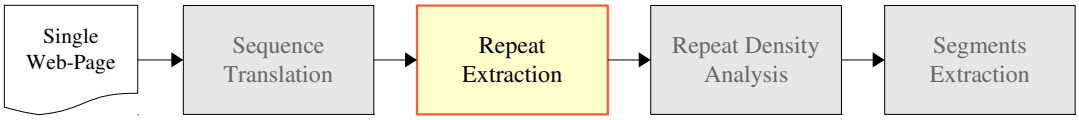


Figure 3.6: *Extended token-based sequence representation*

The differences between the simple and the extended translation should now be more clear. The resulting *simple token-based sequence* (3.5) has strictly one sequence element per html tag (even in the case of the anchor tag <a …> which has 2 parameters) and only one w_{span} element corresponding to the *"Hello World!"* text. In contrast, the resulting *extended token-based sequence* (3.6) breaks up the anchor tag into 3 elements: 1 for the actual tag and 2 for its parameters. Additionally the *"Hello World!"* text now corresponds to 2 w elements, one for each word.

3.3.2 Repeat Extraction



It has already been mentioned many times that the main aim of this project is to assess whether repeat analysis techniques are a suitable tool for identifying structurally similar sections of web-pages corresponding to their content segments. The repeat extraction phase of the system acts as the first step towards this direction.

The sequence translation phase of the system, described above, transforms the web-page's source into a sequence of elements which subsequently becomes the input of the current phase. The type of sequence translation that was used in that step will not be a factor in the repeat extraction phase at all. It will, however, be referred to again in the next operational phase (3.3.3: repeat density analysis).

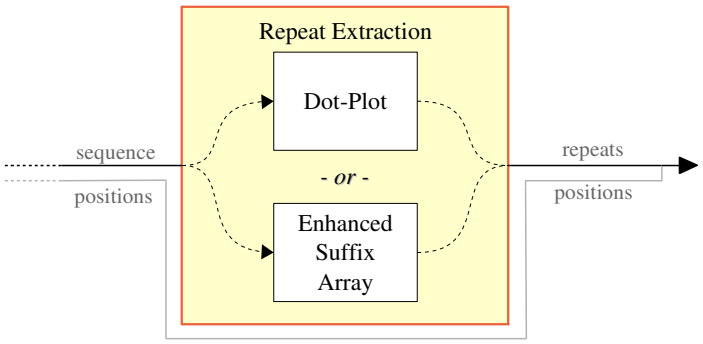


Figure 3.7: *High-level design of the repeat extraction phase*[7]

The repeat extraction phase consists of two major components, which act as alternative ways of extracting repeat information from the inputted sequence (Fig. 3.7). *Dot-Plots* and *Enhanced Suffix Arrays* – both mentioned in Section 2.3 – are well established repeat analysis tools. Assessing their suitability to act as the backbone component for the task of web-page segmentation is the most crucial part of this thesis. In particular, it is expected that the more sophisticated enhanced suffix arrays will provide more fine-grained repeat information compared to the noise-sensitive dot-plots, resulting in better performance (see Chapter 4). Sections 3.3.2.1 and 3.3.2.2 provide descriptions of the two components.

[7]The gray-coloured line arrow labelled as "positions" refers to the position boundaries outputted by the sequence translation component in the case of token-based translation (see 3.3.1). Although not used in the current phase they are necessary for the repeat density analysis component (3.3.3).

3.3.2.1 Utilizing Simple Dot-Plots

The dot-plot is an easy to implement repeat analysis tool. Additionally, it provides a visually interpretable output that can be assessed both by humans and programmatically. On the contrary, dot-plots are known to suffer in cases were the sequences to be analysed tend to be noisy, i.e. containing non-significant repeated characters purely by chance. However, their ease of use makes them a good initial option for assessing whether a sequence has an underlying structure in terms of loosely repeated patterns, which is the type of structure we expect from sequential representations of web-pages to have.

The extraction of repeat information of a sequence from a dot-plot was discussed in Section 2.3.2. It can now be described more formally based on the sequence definitions of Section 3.1. Let S be the input sequence of the repeat extraction phase and $|S| = n$ be its length. By *sliding* the sequence S against itself we create a total number of $2n-1$ possible alignments. For each alignment a number of elements might match in the sliding sequences. Each such match corresponds to a single repeated element c appearing in the sequence at positions i_c and j_c[8]. This process makes sense intuitively when illustrated by visualizing the actual dot-plot. A *visualized* nxn sized dot-plot has a total number of $2n-1$ diagonals, with each diagonal being the visual equivalent of an alignment derived from the sliding sequences. Position pairs of matching characters are marked with a dot in the corresponding coordinates of the plot. Obviously, the main diagonal will always be an unbroken line of dots and the dot-plot will be symmetric with respect to it.

At this point, it should be clarified that in its essence, a sequence's dot-plot refers to the set of position pairs corresponding to repeated elements in the sequence and **not** its visualized representation. However, the graphical illustration of it as a recurrence-type plot makes it visually interpretable and easier to understand. Although the actual output of the system's dot-plot component is in terms of position pairs, we will consistently use its visual representation, both for this component, as well as for the enhanced suffix array component mentioned in the following section.

The example provided in Fig. 3.8 illustrates the construction of a dot-plot. A character-based sequence of a toy web-page is used (and will be used throughout the chapter for clarification purposes). In the upper part of the figure one of the possible alignments of the sequence with itself is shown. In the current alignment, there exist 4

[8]The formal definition of the repeated pair positions would be $R = \langle (i_c, i_c), (j_c, j_c) \rangle$. For simplicity reasons we only refer to the pair using i_c and j_c since it is unit-length.

matched elements (coloured red). Their corresponding dots in the plot are also shown in red colour. Note that there are 8 red points, 4 in each side of the main diagonal in symmetrical fashion. Each group of 4 red dots are part of the same diagonal since they come from a single alignment. The rest of the plot is constructed similarly.

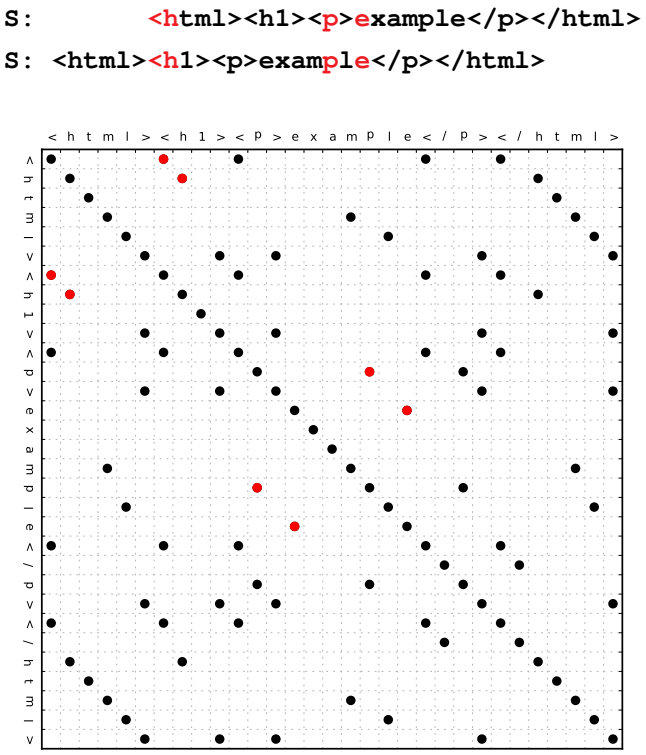


Figure 3.8: *Example of dot-plot visualization from a toy page sequence*

Dots appearing at neighbouring coordinates in the same diagonal indicate that more than one consecutive elements are repeated. For example the subsequence $\omega =$ "html>" which appears at positions $R = \langle (1,5),(26,30) \rangle$ can be seen as a 5-dot long diagonal near the upper right and bottom left corner of the dot-plot. Such repeated patterns are easily identified by human inspection but dot-plots provide no efficient way of extracting them as one single repeated sub-sequence. In addition, their quadratic search space makes greedy approaches infeasible.

This inability to efficiently distinguish between significant repeated sub-sequences and short – or even unit-length – ones, whose existence is probably irrelevant to the task they're applied to, is the most prominent limitation of dot-plots. In the following section we describe how this limitation can be addressed by using a more sophisticated repeat analysis tool, the enhanced suffix array.

3.3.2.2 Utilizing Enhanced Suffix Arrays

Suffix arrays and the more sophisticated extension, the enhanced suffix arrays, are data structures of huge importance in sequence and repeat analysis. They allows for extremely efficient solutions – both time- and space-wise – to numerous sequence related problems and are in many cases the core component of complex sequence mining systems (e.g. the genome analysis REPuter system [Kurtz et al., 2001]). They lie in the opposite side of dot-plots with regards to the trade-off between intuitive simplicity and performance. They provide a highly complex but more usable and flexible method of extracting repeat information.

The specific details regarding their structure, and the algorithms that utilize them are not in the scope of this project[9]. Instead, a high-level description of their capabilities is given below.

In terms of input and output the (enhanced) suffix array component works in the same way as the dot-plot, which makes their use as alternatives for the phase of repeat extraction very convenient. The input sequence is used to construct the main building blocks of the data structure. These building blocks are essentially a number of arrays that provide – among others – easy access to the sequence's suffixes in lexicographical order, as well as information about these suffixes' longest common prefixes. Efficient algorithms are able to utilize this information in order to extract repeats. An over-simplified example is shown below to give some insight of the main idea behind suffix arrays.

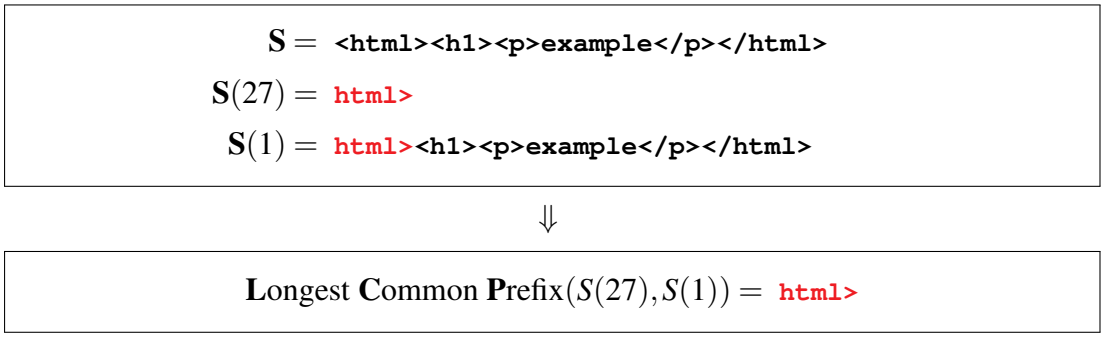


Figure 3.9: *Simplified example of repeat identification using suffix and prefix information*

[9] See the paper that introduced the Enhanced Suffix Array [Abouelhoda et al., 2004] for full details.

Although simpler from the actual repeat extraction algorithms, Fig. 3.9 shows how the longest common prefix of lexicographically ordered suffixes $S(27)$ and $S(1)$ are in fact repeated sub-sequences of the original sequence S.

An important advantage of enhanced suffix arrays in comparison to simple dot-plots is their ability to extract whole repeated sub-sequences at once, instead of the single repeated elements that are outputted by dot-plots. More specifically, given an enhanced suffix array constructed from an input sequence, there exist algorithms for extracting the sequence's maximal and/or supermaximal repeated pairs. Even more importantly, it is trivial to threshold the extracted (super)maximal repeats both in terms of their length as well as of the number of times that each repeat is found in the sequence. This acts as a major improvement over the naive repeat extraction method of dot-plots, since one can now specify how significant a repeat should be in order to be extracted. Too short or not frequent enough repeats can be ignored by setting these two thresholds which will from now on be referred to as *minlen* and *minrep* parameters respectively.

In order to make the effect of these parameters clearer, a series of example figures will be used. For this purpose we will once again use the toy character-based sequence used previously.

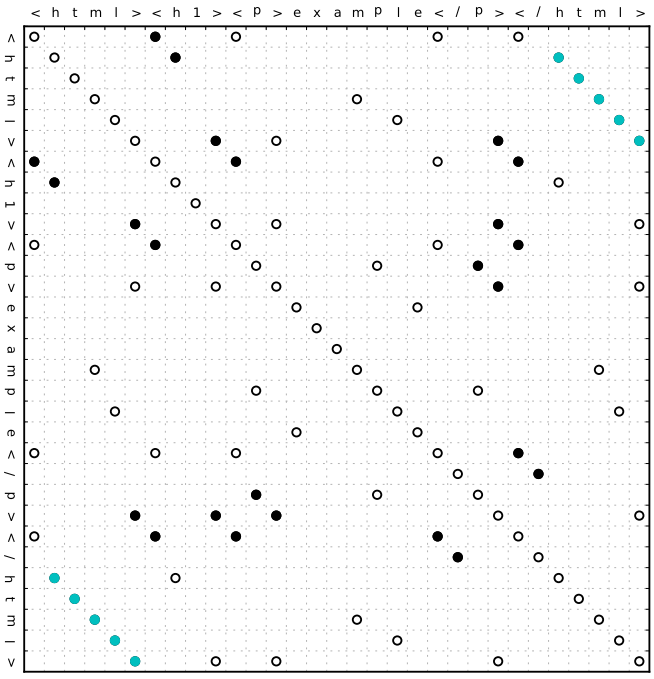


Figure 3.10: *Example output of the dotplot component with varying dot colours illustrating repeat significance*

Firstly, the output of the dot-plot component, which was also illustrated in section 3.3.2.1 is shown below in Fig. 3.10. As noted earlier, diagonally consecutive dots indicate repeated sub-sequences of length greater than 1. For clarity reasons, the dots in Fig. 3.10 are coloured based on the length of the repeats they correspond to. Single repeated elements are coloured in white (as is the main diagonal since it is redundant), repeats of length 2 are coloured in black and the only repeated pair of length 5 is coloured in cyan. In contrast to the inability of the dot-plot component, the enhanced suffix array component is capable of extracting only the portion of these repeats that satisfy the *minimum length* and *minimum repetitions* parameters.

Suppose that only repeats of length greater than or equal to 2 elements need to be extracted. By setting the minimum length parameter to $minlen = 2$ and the minimum repetitions parameter to its default value $minrep = 2$, only the repeats shown in Fig. 3.11(a) are extracted. Similarly, by applying an even more strict threshold for the minimum length of extracted repeats ($minlen = 5$), only one repeated pair is extracted 3.11(b).

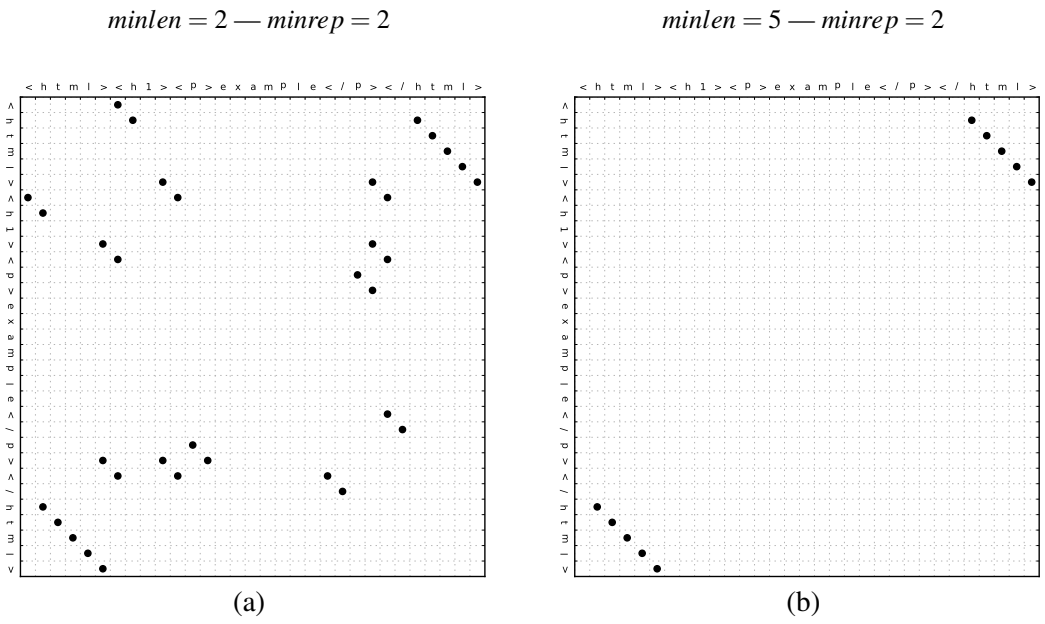


Figure 3.11: *Length-thresholded repeat extraction using enhanced suffix arrays*

It is similarly simple to threshold the number of times each extracted repeat is found in the original sequence by setting the minimum repetitions parameter to $minrep = 3$, which requires for a sub-sequence to be repeated a minimum of 3 times in order to be

extracted. Once again a $minlen = 2$ is required. As illustrated in Fig. 3.12(a) only the sub-sequence "><" appears 3 times in the original sequence. If, however, the length parameter is set to $minlen = 5$ no repeats exist meets the required thresholds (Fig. 3.12(a)).

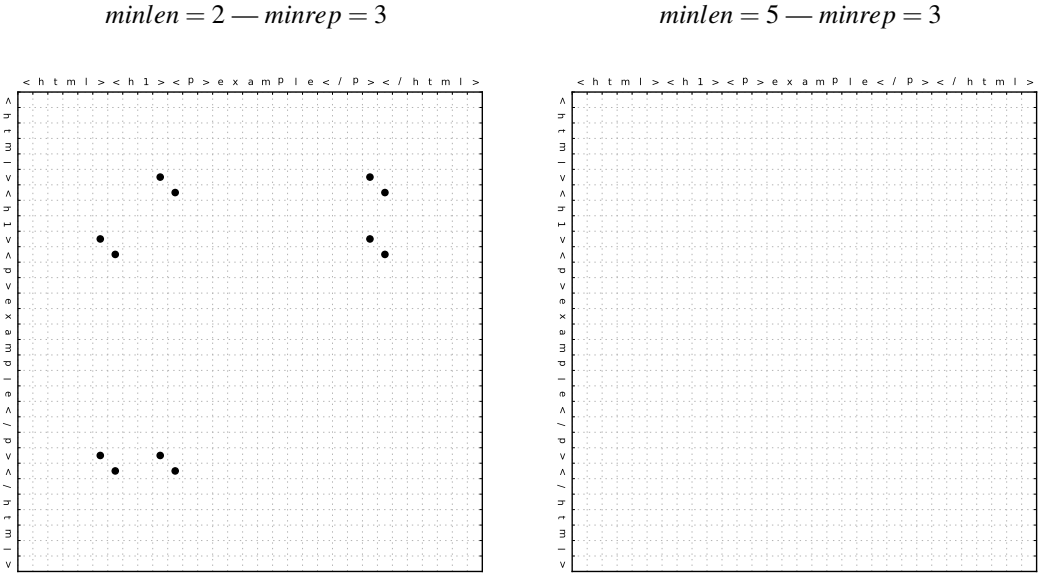


Figure 3.12: *Length- and frequency-thresholded repeat extraction using enhanced suffix arrays*

As with the dot-plot component's output, the enhanced suffix array component – given the appropriate parameters – outputs a list of repeated pairs $R = \langle (i_1, j_1), (i_2, j_2) \rangle$ for each repeated sub-sequence. The output is illustrated in a similar way dot-plots do in order to be more easily interpreted. By examining the aforementioned examples it should be clear that the enhanced suffix array component has the appropriate mechanisms to cope with noisy sequences, while the dot-plot component does not. Chapter 4 will focus – among others – to assessing the difference between the two components.

3.3.3 Repeat Density Analysis

The repeat extraction phase of the system was responsible for extracting and outputting repeated pairs from the input sequence representation of a web-page. Obviously repeats of arbitrary length and frequency exist in abundance in long and complicated sequences of elements as the ones our system dealing with. The main idea on which this web-page segmentation system is based, is identifying regions of the web-page sequence representation that exhibit a high repetitiveness rate. In this section, the repeat density analysis component, which is responsible for this task, will be described.

Each repeated pair $R = \langle (i_1, j_1), (i_2, j_2) \rangle$ extracted from the previous phase can be interpreted as follows: all sequence elements with positions $p : p \in (i_1..j_2) \cup (i_1..j_2)$ are part of a repeat. Given a list of repeated pairs, each element $S[p]$ of the sequence will be part of 0 or more repeats. Due to the numerous repeats that are evident in each sequence, the number of repeats that each element is a part of is not expected to follow a smooth enough distribution, in order to be analysed in an automated way. Our goal is to deal with the erratic behaviour of repeat distribution by applying a smoothing technique to this repeat distribution.

More specifically, the *repeat density analysis* phase can be broken down to the three following steps:

1. Gather repeat information for all elements in the sequence. This is done using the outputted repeated pairs of the repeat extraction phase.

2. Summarize the number of repeats at each position, i.e. for each position in the sequence count the number of repeats that it is a part of. This will result in a histogram-like summary where each element of the sequence is associated with its repeat count.

3. Smooth the resulting histogram in order to get a continuous distribution indicating the estimated *density* of repetitions for the whole sequence.

The first two steps are trivial. The outputted list of repeated pairs from the previous phase can be directly used for this purpose. In terms of a dot-plot representation, the number of repeats that each element is a part of can be identified by counting the amount of *dots* that appear on each row of the plot. This way, the histogram-like summary of the number of repeats per position is obtained by simply adding the number of dots in each row. This can be done regardless of whether the extracted repeats where generated by the dot-plot or by the enhanced suffix array component.

Regarding the third step, in order to smooth this distribution of repeat counts, a *kernel-based density estimation* (KDE) technique named *kernel smoothing* is used. Kernel smoothing is a non-parametric method where a real-valued function f is estimated using noisy observations [10]. This results in a real-valued repeat density estimation that is more suitable for further analysis in the segments extraction phase of the system. A very simple illustration, which is conceptually similar to our use of a kernel smoothing technique is shown in Fig. 3.13, where the real valued function drawn with a red line is estimated using a number of observations represented as a histogram.

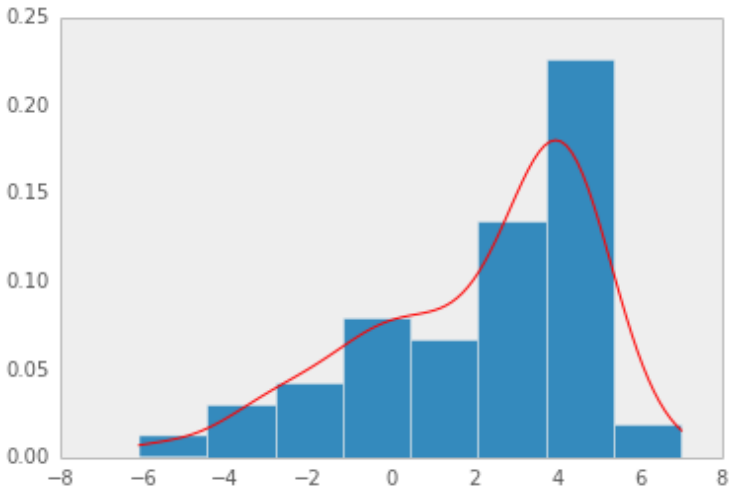


Figure 3.13: *Simple example of kernel smoothing*

[10] For a summary of smoothing techniques see `http://data.princeton.edu/eco572/smoothing.pdf`

Once again, we provide a number of illustrative examples using the same toy character-based sequence as before. In the following figures the process of transforming the repeat extraction output of the previous phase into a histogram-like summary and subsequently into an estimated density distribution is shown. Examples use both the dot-plot component's output as well as outputs from the enhanced suffix array component for different parameter settings.

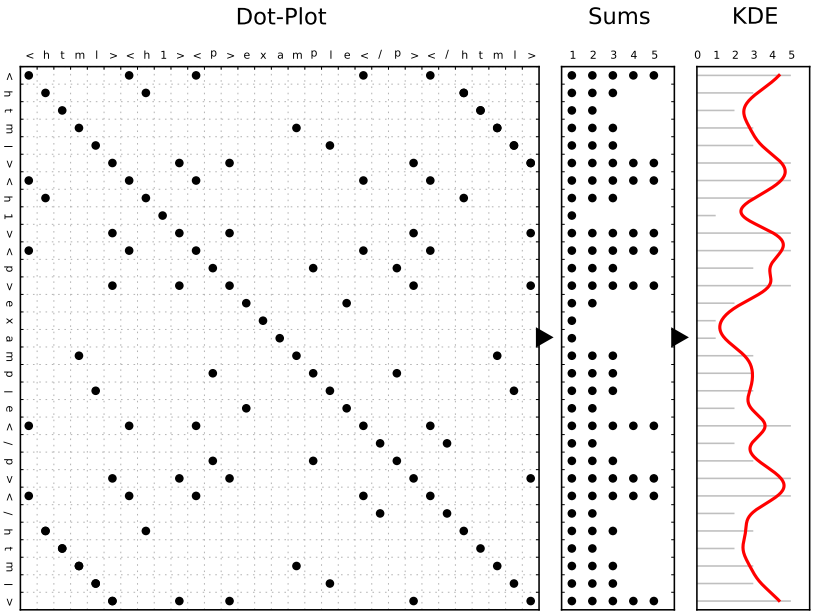


Figure 3.14: *Repeat density analysis of repeats extracted from the dot-plot component*

In Fig. 3.14 the input of the repeat analysis phase is the list of repeated elements extracted from the dot-plot component (displayed visually in the leftmost part of the figure). The repeat summarization step is performed by adding the number of dots per row, which results in the histogram-like representation shown in the middle part of the figure (rotated vertically for clarity reasons). The erratic behaviour of the histogram is evident in this case. As a solution to this problem, the kernel-smoothing step of the process is applied to the data and results in the continuous distribution shown in the rightmost part of the figure (red line).

The same process is illustrated in Fig. 3.15, only this time the repeats that are fed into the density analysis phase are extracted by the enhanced suffix array component. The *minlen* and *minrep* parameters used for repeat extraction in each figure are noted above the corresponding dot-plot.

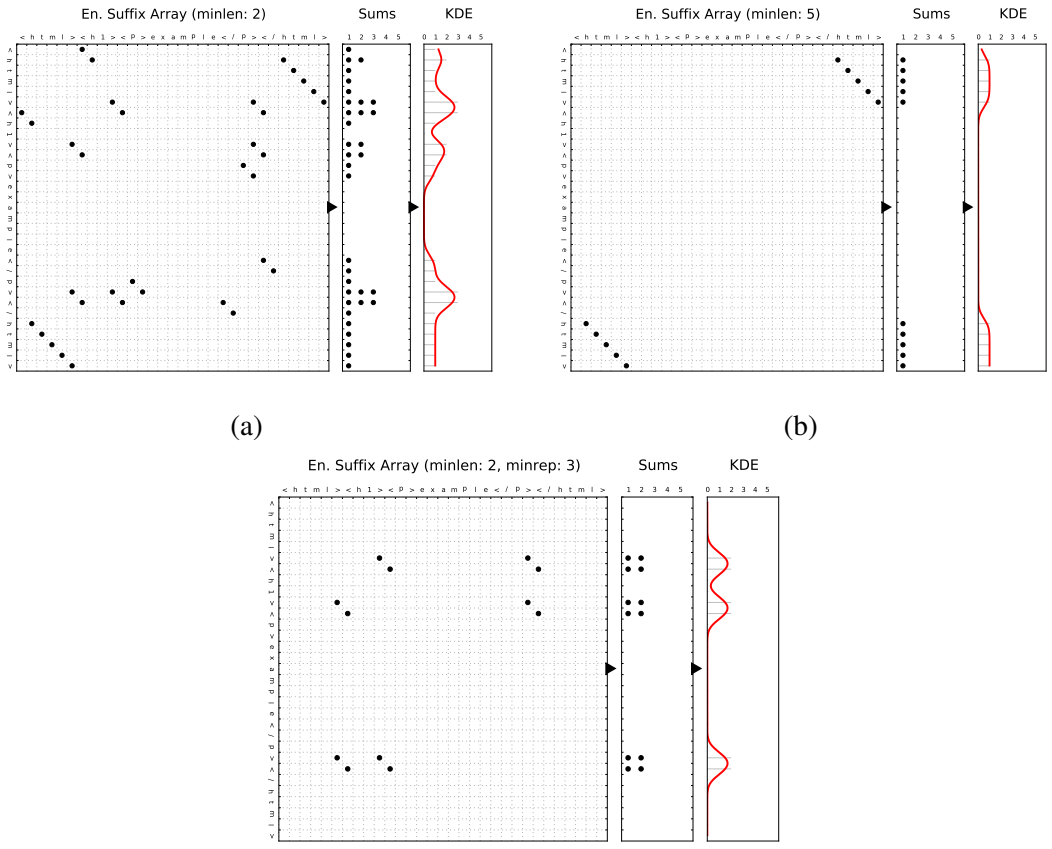


Figure 3.15: *Repeat density analysis of repeats extracted from the enhanced suffix array component*

Although this is a toy example, it is easily noticeable that the repeat density distributions resulting from the enhanced suffix array component are more well-behaved and regions of high repeat density are more easily identified.

All the examples above have been dealing with a character-based sequence representation. As described in section 3.3.1, the two alternatives of the simple and extended token-based sequence representations also exist. Although the repeat extraction is performed on the respective sequence representation (character- or token-based) the repeat density analysis is always done on a character level. This is made possible using the position boundary pairs that are outputted from the sequence translation component (see Fig. 3.5 and 3.6).

Recalling what was mentioned earlier, each sequence element resulting from a token-based translation corresponds to a starting and ending position pair that maps the translated element to the characters of the source it was derived from. Using these boundaries makes it trivial to map the extracted repeats from a token-based sequence back to the original source characters they correspond to. Any *dot* indicating a repeated

token-based element will be mapped to the consecutive characters-based elements it is derived from, incrementing all their repeat counts by 1. This means that the histogram and density distribution produced by steps 2 and 3 of the repeat density analysis phase respectively will always be in terms of the original character-based sequence.

3.3.4 Segments Extraction

The repeat density analysis phase of our system produces a density distribution that estimates the number of repeats that correspond to each element of the character-based representation of a web-page. Our hypothesis is that regions of the web-page that are densely populated by repeats, are good estimations of where the segments to be extracted exist.

It is now trivial to identify those regions of high repeat density given the output of the previous phase. This is achieved by applying a simple peak extraction algorithm that identifies any local maxima that exist in the repeat density function. A simple example of peak identification is shown in Fig. 3.16. The repeat density function illustrated is the one produced by the example of Fig. 3.15(a) shown in more detail and horizontally oriented. The extracted peaks are marked with blue **x**'s[11].

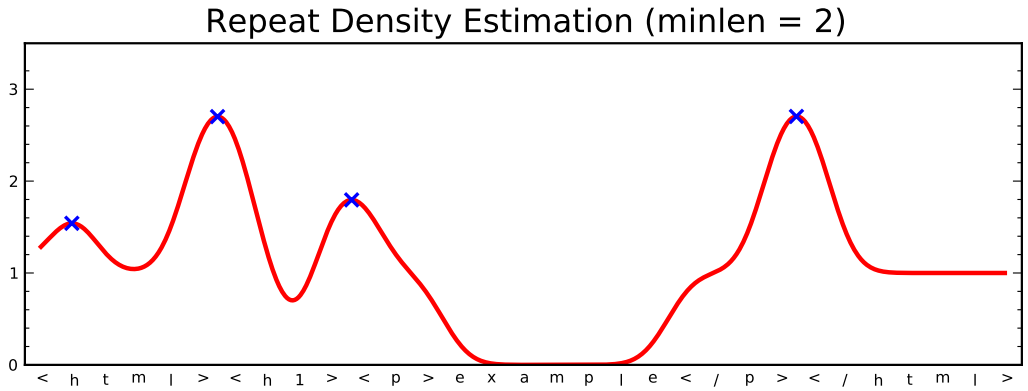


Figure 3.16: *Peak extraction applied to a repeat density function*

[11] Even in this toy example, the identified peaks are placed close to areas where opening and closing html tag brackets exist, which are actually the most common elements of the sequence.

We expect that these peaks are – loosely – positioned at the center of the content segments to be extracted. Given a set of peaks, the second part of the *segment extraction* phase constitutes of two alternative approaches of extracting those segments. The first, a naive one, makes no discrimination between peaks, while the second one utilizes a clustering algorithm in order to only extract segments around a subset of peaks. Both methods are described in the sections below.

3.3.4.1 Naive Extraction

Recall the *interspersed* segmentation definition provided in section 3.1.3. According to that, any sequence S of length n is associated with an M-length array $\langle p_1, p_2, \ldots, p_M \rangle$ of *potential boundary positions*, which specify $M+1$ adjacent non-divisible elements $\langle [0, p_1), [p1, p_2), \ldots, [p_M, n] \rangle$.

In the case of web-page segmentation and, particularly, the character-based sequences that we always deal with at this phase of the system, it makes intuitive sense to set these M potential boundaries at the starting and ending positions of html tags. By doing so, we make sure that no html tag is split in half by a boundary and that any content section in-between html tags always remains undivided.

Given the M-length array of potential boundaries described above, the naive segment extraction method is performed in a straightforward manner. For each peak extracted from the repeat density function, we identify the potential boundary position that is positioned closest to it. This is considered the *"anchor"* of the current segment and will be referred to as $p_a \in \langle p_1, p_2, \ldots, p_M \rangle$.

Now, our goal is to extend p_a to both directions in order to obtain the appropriate opening (p_{open}) and closing (p_{close}) boundaries for the current segment. We do this in a trivial way, by incrementally advancing the candidate boundaries to the left and to the right respectively by one boundary position at a time. This incremental procedure is terminated either when we reach a position that exceeds the middle point between the current and the next/previous peak, or when a maximum allowed margin between p_{open} and p_{close} is reached[12]. By doing so for every peak, we end up with a set of segmentation boundaries which are represented as an *interspersed* segmentation array $\ddot{\mathbf{X}}$ – as defined in 3.1.3.

[12]This maximum margin is empirically learned and is set to 1.5 times the average true margin of any segment in the training instances.

An example of a final segmentation output of our system using the naive segmentation is shown in Fig. 3.17. In it, the repeat density distribution of a single web-page – from the youtube.com user videos web-source – is drawn using a red line. The ground truth segmentation of the particular page is illustrated using the vertical green regions bounded with black lines indicating the segmentation boundaries[13]. The predicted segments of our system are marked using horizontal magenta lines indicating their spanning area.

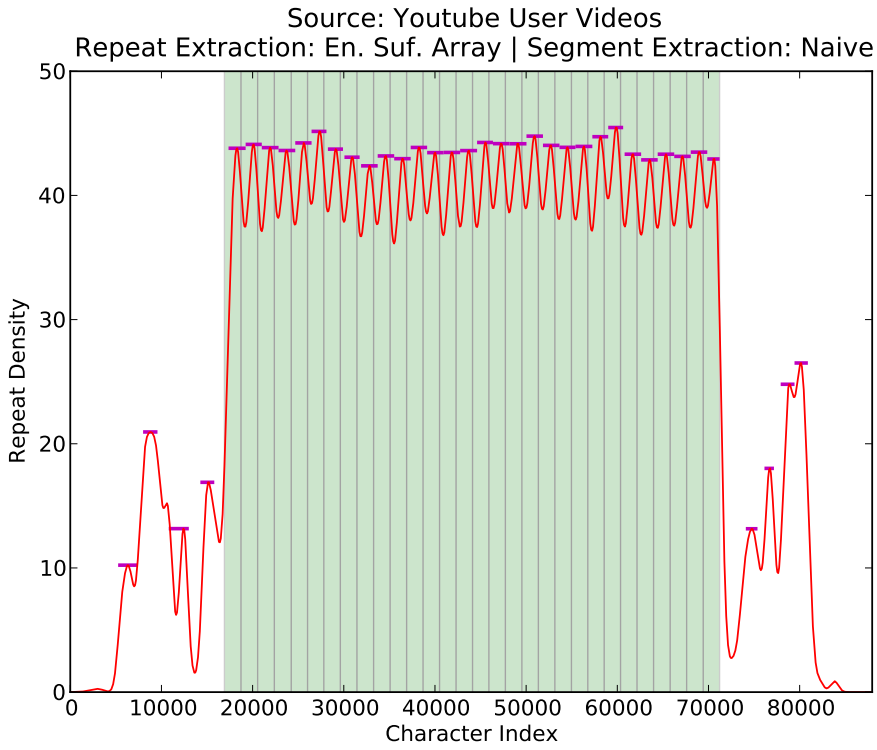


Figure 3.17: *Example segmentation output with naive extraction*

The issue with this naive approach is evident in the above graph. Although a very precise identification of the true segments is achieved, our predicted output includes a number of segments that do not correspond to any actual content section. This is due to the abundance of repeated patterns in the sequence that may not necessarily be related to the segments we want to extract.

[13] In this case the segments happen to be adjacent.

3.3.4.2 Extraction via Peak Clustering

A simple, yet effective concept for improving the naive segmentation extraction component described above has been designed as an alternative for this system phase. As seen on Fig. 3.17, not all peaks corresponded to the actual content segments we wanted to predict. However, the peaks that did correspond to the content segments exhibited a homogeneity in terms of their height. The idea behind this segment extraction technique is to group all similar, in terms of height, peaks into peak subsets. Then the most dominant subset could be selected and only segments from its peaks will be extracted.

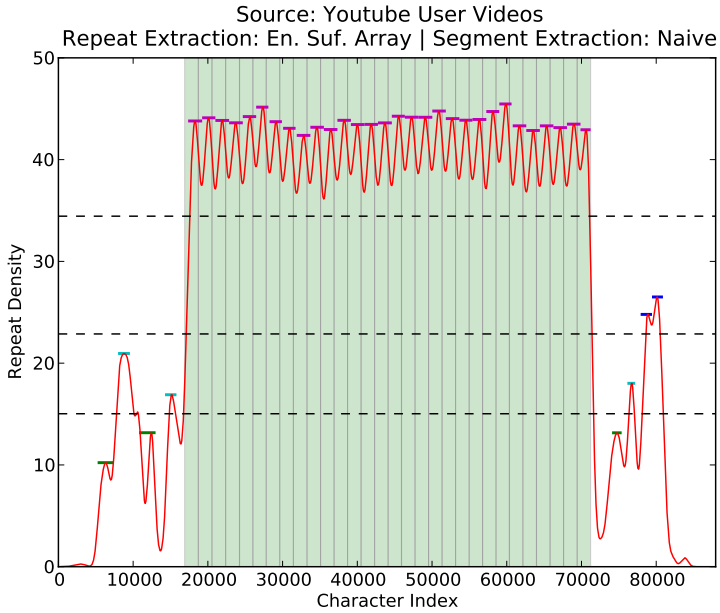
The obvious choice for any task involving the grouping of similar instances is clustering techniques. The simplest clustering algorithms, *k-means*, has the drawback that requires the number of groups evident in the data to be known beforehand, which is not the case for our problem.

The alternative to k-means is the more sophisticated family of methods referred to as *hierarchical clustering methods* and in particular the *single-link agglomerative clustering algorithm* [Sibson, 1973]. In single-link agglomerative clustering, each observation is initially assigned to its own group/cluster. Subsequently, pairs of clusters are merged in a bottom-up manner based on which two clusters include the two most similar observations. As a result bigger clusters will start to form. The procedure is continued until a cut-off condition is met.

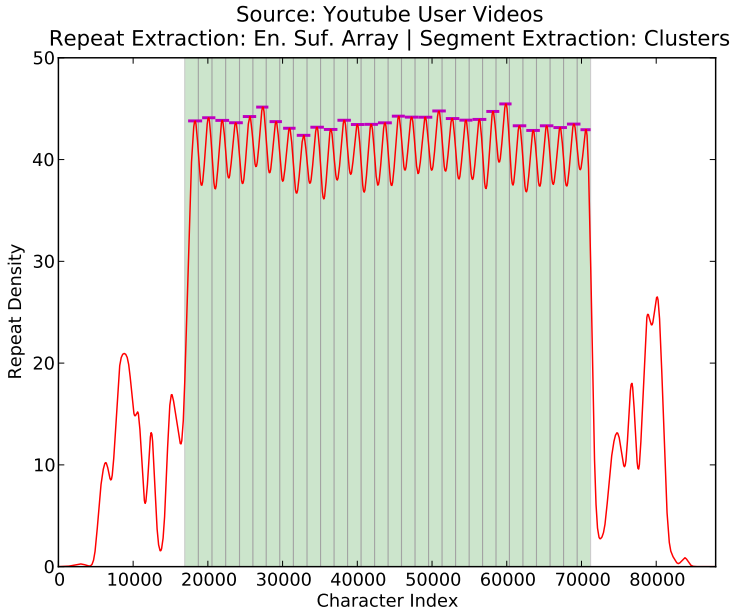
In our context, each peak is an observation with a single feature; its height. Using the above described procedure, clusters of peaks with similar heights will begin forming groups. The process will be terminated when the next cluster merge that is about to occur would be between two clusters whose height distance is above an empirically chosen threshold[14].

The output of the above algorithm is a set of clusters, each consisting of an arbitrary number of peaks, as shown in Fig. 3.18. The dashed horizontal lines roughly indicate the decision boundaries that split the clusters (in terms of peak heights). We continue by extracting the segments that correspond to each peak in the same way we did for the naive method. The colour of the segment marks in Fig. 3.18(a) indicate which segments are grouped together.

[14]The threshold is set to $t = 0.1$ x maximum height difference between any two peaks.



(a)



(b)

Figure 3.18: *Illustration of peak clustering (a) before and (b) after cluster selection*

The next step of our method is to decide which cluster should be selected to output the final segmentation prediction. In order to do so we need to define the total significance of the segments in each cluster. We expect that the actual content segments will span the largest portion of the web-page. Based on that, we select the cluster that spans the largest portion of the web-page in terms of characters in-between html-tags.

Table 3.1 shows the content character counts for each cluster of the example webpage.

Cluster (Colour)	**Content characters**
1 (Magenta)	2188
2 (Blue)	84
3 (Cyan)	188
4 (Green)	230

Table 3.1: *Total number of content characters in each cluster*

By selecting the cluster that spans the largest overall area in terms of in-segment content characters, our expectation is that in most cases the correct cluster will be selected. In the example of Fig. 3.18 the resulting segment extraction is an obvious improvement compared to the naive method.

Chapter 4

Experiments

The system that was designed and built for this thesis consists of 4 main building blocks, as those described in chapter 3. Each of these operational phases has a number of major variations and/or tuning parameters that are expected to affect its performance.

Some of these variations/parameters are either easy to be tuned and result in non-significant performance changes or, in some cases, their trivial nature is simply not interesting enough to require proper experimental assessment. Therefore, the focus of our experimental evaluation efforts is shifted towards those factors that constitute the design backbone of our system and any parameters that control their operation.

Having this in mind, the evaluation phase of this project was designed in order to experimentally assess the following hypotheses:

1. Repeat analysis techniques and tools can be utilized for the task of web-page segmentation in the context of content extraction. The two basic components for repeat extraction that were used (dot-plots and enhanced suffix arrays) will be tested and compared in order to verify this claim.

2. The sequential nature of web-pages provides a very flexible basis for repeat analysis, since different levels of sequential abstraction can be used with varying results. Three different sequential representations will be evaluated.

3. The segment extraction phase of our system utilizes the novel process of repeat density analysis in order to output the predicted web-page segments. This is achieved using two segment extraction methods: a naive one and a more sophisticated which uses a clustering technique. The potential improvement that the clustering method brings will be assessed.

The rest of this chapter is structured as follows: In section 4.1 the dataset that was used for the experimental evaluation will be described. Section 4.2 will provide the basic methodology of the evaluation in terms of the objective function that was used and any secondary evaluation metrics utilized. Additionally, a brief description on how the ground-truth for these experiments was established will be given. Finally, section 4.3 will provide detailed descriptions on the experiments performed and their outcome, as well as critical discussion of these results.

4.1 Dataset

One of the most significant issues with previously developed web-page segmentation techniques was the insufficient focus that was put upon creating a diverse enough benchmark dataset, in order to train, test and evaluate the segmentation methods. For this reason, one of the main objectives of this project was to create an appropriate dataset that will be used for training and evaluating the performance of the built project and – possibly – of future efforts on web-page segmentation in this context.

The dataset to be created will consist of a large number of web-pages as html source code files. The first step towards building this dataset is to identify a diverse enough set of web-domains that are suitable for the task of web-page segmentation. The task that this project deals with sets a few requirements on the type of web-domains that should be used. Namely, the key points to be met are:

- All web-pages should have a group of multiple content sections of similar structure.
- The sections should exhibit adequate level of visual and content homogeneity.
- The sections should be presented in a list- or grid-like manner that makes their grouping obvious.
- The total area of the web-page covered by the content sections should be large enough to be considered the most significant part of the page.
- Gathering a large number of web-pages from the domain should be fairly easy and not violating the web-domain's crawling policies.

Luckily, an abundance of web-domains that meet these requirements can be found in the WWW today. Online stores, search engines of any type, blog-like web-pages,

user communities, video streaming services are just a small portion of the kind of sources we should be looking for.

After careful examination of various web domains a total number of 7 different sources were selected coming from 5 web-domains so that the aforementioned criteria would be met and the resulting collection would be as diverse as possible. The 7 sources are presented in Table 4.1:

Web-domain	**Web-site Description**	**Source Type**	**Presentation**
google.com	Search Engine	Search Results	List
bing.com	Search Engine	Search Results	List
reddit.com	Discussion Community	User Posts	List
ebay.co.uk	Online Store	Search Results	List
			Grid
youtube.com	Video Streaming Service	Search Results	List
		User Videos	Grid

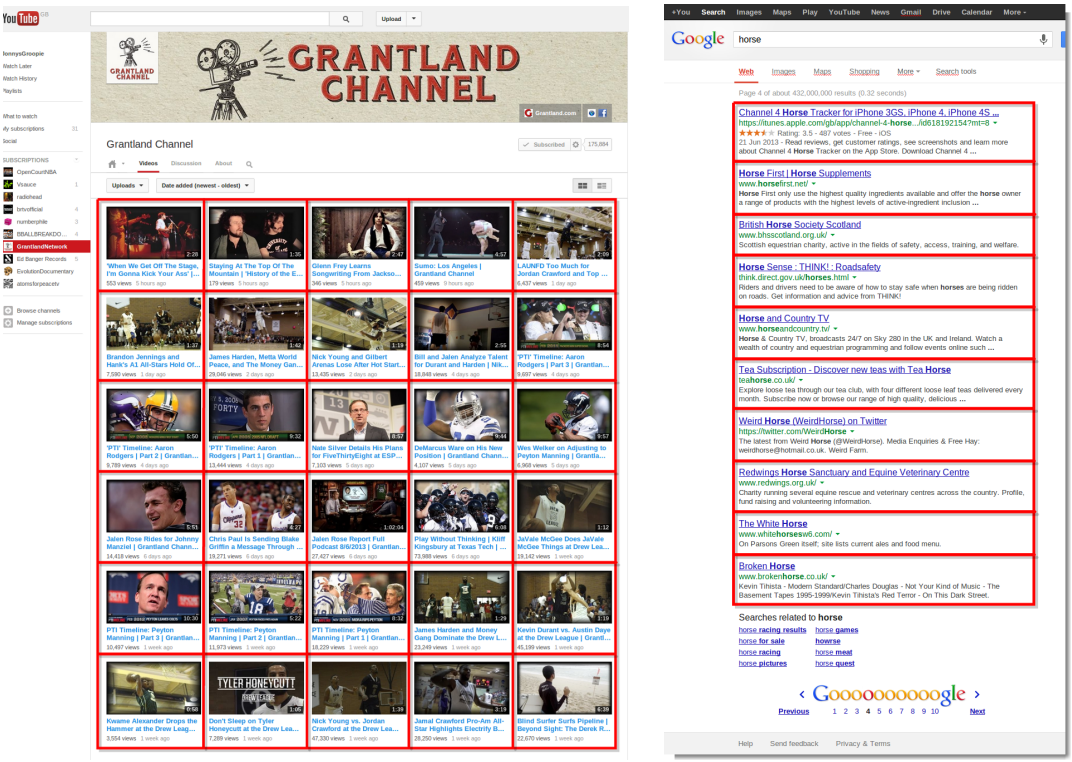
Table 4.1: *Description of web-page sources used in the dataset*

Custom-built web-crawlers were developed in order to automatically obtain a large number of individual web-pages from each source, while respecting the crawling policies of each web-domain. A diverse list of short queries was created in order to obtain pages from the 4 sources that consist of query-based, search results web-pages (see Table 4.1). The rest of the sources wer crawled in the usual, url-following manner. Statistical summaries of the pages fetched from each source are given in Table 4.2:

Source	**# of Pages**	**# of Segments/Page**[1]
google.com	1920	10
bing.com	1920	10
reddit.com	1903	25
ebay.co.uk (List View)	1822	25/50/100[2]
ebay.co.uk (Grid View)	1825	24/48/96[2]
youtube.com (Results)	1800	20
youtube.com (User Videos)	1663	30
Total	**12853**	–

Table 4.2: *Dataset statistics*

As Tables 4.1 and 4.2 show, the resulting dataset covers a large spectrum of web-page styles in terms of web-domains, types of data presented, size of content sections and presentation layout. Figure 4.1 shows two representative, but very different from each other, web-pages from our dataset. In Fig. 4.1(a) a grid-layout, strictly structured web-page, containing 30 content sections, each of them covered mainly by a picture is provided. In Fig. 4.1(b) a list-layout, loosely-structured web-page, containing 10 content section, each of them consisting purely by text is shown.



(a) Grid Presentation (youtube.com) (b) List Presentation (google.com)

Figure 4.1: *Examples of web-pages from our Dataset*

4.2 Evaluation Methodology

The segmentation system built for this project has many unique properties in terms of the novelty of techniques that it utilizes and combines for the purpose of web-page segmentation. However, our goal had always been not to develop a segmentation algorithm in a heuristic manner, but instead, learn how to segment web-pages adaptively,

[1]Refers to the typical number of content sections per page. Variations may exist.

[2]Ebay.co.uk allows control over the number of results per page. All 3 given option were used to achieve increased data diversity.

based on existing data. For this purpose, our segmentation system has been built following the basic principles of machine learning. It has been repeatedly reported in this thesis – and formally described in chapter 3 – that each pipeline component has a number of variations or parameters that are expected to affect the overall performance. Instead of deterministically deciding the settings of the system based on our intuition or on empirical observations, any parameter that is expected to be an important factor on the segmentation output will be learned in a principled way. In order to do so, two key decisions need to be made:

- How will the ground-truth segmentation of our dataset be generated, and
- Which objective function will be used to learn any significant component parameters and to compare major component variations.

4.2.1 Ground Truth Generation

The need for a ground truth in most machine learning algorithms has a dual utility. Firstly, it is used in the training procedure of the system in order to optimize any parameters and, secondly, it is used as a reference in the testing procedure in order to assess the overall system performance or to compare major operational variations.

In the context of our task, the ground truth segmentation of a web-page is defined as follows. Let S be the character-based sequence of a web-page of length n. Let $\langle p_1, p_2, \ldots, p_M \rangle$ be an array of *potential boundary positions* in S. Then, $\ddot{\mathbf{T}}$ is an M-length array defining the ground truth *interspersed* segmentation of S, such that all subsequences of S that are defined as segments in-between opening and closing boundary positions in $\ddot{\mathbf{T}}$ correspond to actual content sections of the web-page.

The manual extraction of the true content section of the web-pages in our dataset is a tricky task. It requires careful inspection of the pages' source code from all the domains in order to identify the exact patterns that correspond to the starting and ending positions of each segment. Because of the structured nature of multi-content web-pages there is almost perfect consistency throughout each source in our dataset regarding the sequence of html tags that denote these starting and ending positions.

By compiling the appropriate regular expressions for each source and and running segmentation scripts for all web-pages of the data-set we obtain a set of ground truth segments $\ddot{\mathbf{T}}$ – as it was described above – for each page.

The non-trivial procedure of manually segmenting these web-pages underlines the importance of a web-page segmentation system like the one proposed in this thesis.

4.2.2 Evaluation Metrics

In section 2.4, a brief description of the family of *window-based* evaluation metrics, designed for segmentation tasks, was given. In this section, formal mathematical definitions of two of these metrics will be provided[3]. An important issue that makes these metrics unsuitable for the evaluation of our task will be explained. Finally, an extension of these metrics that accounts for this issue will be proposed as an objective function for our experiments. For ease of understandability, the two segmentation definitions given in 3.1.3 will be re-introduced.

P_k and k-κ

The most common way to formally define a segmentation task is the following[4]. Each sequence S of length n, which is to be segmented, is associated with an M-length array $\langle p_1, p_2, \ldots, p_M \rangle$ of *potential boundary positions*, which specify $M+1$ adjacent non-divisible elements $\langle [0, p_1), [p1, p_2), \ldots, [p_M, n] \rangle$.

A segmentation $\mathbf{X}$ is defined as a sequence of Boolean variables $\langle X_1, X_2, \ldots, X_M \rangle$, such that $X_m = 1$ if there is a boundary at p_m, and $X_m = 0$ otherwise. Let $\mathbf{X}_i^j$ denote a subsequence of $\mathbf{X}$ from i to j inclusive, referred to as a *window*, and let $\Sigma\mathbf{X}_i^j$ denote its sum. Finally, let $\delta(\mathbf{X}, i, k)$ be the Boolean *boundary presence indicator function*, indicating whether a boundary is present in a k-length window starting at i:

$$\delta(\mathbf{X}, i, k) = \begin{cases} 0 & \text{if } \Sigma\mathbf{X}_i^{i+k-1} = 0 \\ 1 & \text{otherwise.} \end{cases}$$

Let $\mathbf{R}$ and $\mathbf{H}$ be a reference and a hypothesized segmentation respectively. We define the segmentation evaluation metric $P_k(\mathbf{R}, \mathbf{H})$ as:

$$P_k(\mathbf{R}, \mathbf{H}) = \frac{\sum_{i=1}^{M-k+1} \mathbf{1}[\delta(\mathbf{R}, i, k) - \delta(\mathbf{H}, i, k)]}{M - k + 1} \tag{4.1}$$

where k denotes the *window length* and $\mathbf{1}[x]$ is a non-zero indicator function, evaluating to 1 if $x \neq 0$, and 0 otherwise. The conventional value of $k = max(\lfloor (M+1)/2((\#of\ ref.\ segments) + 1)) \rfloor$ was used.

[3] This definition strictly follows the one found in Niekrasz and Moore [2010].

[4] Note that this is **not** the definition that applies in our for our task. It is provided in order to make the metrics understandable. See rest of the section for how these metrics are extended for our purposes.

In simple terms, the $P_k(\mathbf{R},\mathbf{H})$ metric calculates the proportion of k-length windows, where **R** and **H** disagree about the existence of a boundary[5]. A P_k value of 1 denotes complete disagreement, while a value of 0 denotes perfect agreement between **R** and **H**.

Although P_k is widely used as a benchmark segmentation metric, Niekrasz and Moore [2010] concluded that it is positively biased towards hypothesized segmentations with either very few boundaries or boundaries mainly concentrated around the edges of sequences.

Furthermore, they propose the novel k-κ metric that deals with these biases by accounting for chance agreement. k-κ is defined as:

$$k\text{-}\kappa(\mathbf{R},\mathbf{H}) = \frac{1 - P_k(\mathbf{R},\mathbf{H}) - C}{1 - C} \tag{4.2}$$

where C is the probability of having an agreement between **R** and **H** by chance. A k-κ value of 0 reflects chance agreement, a value of 1 is perfect agreement, and a value of -1 is complete disagreement.

The proposed $\ddot{P}_k$ and $\ddot{k}$-κ

Both metrics mentioned above are designed to evaluate segmentations where a sequence is segmented into adjacent segments that cover the whole sequence. However in our task, the segments to be extracted are not guaranteed to be adjacent – but may as well be –, since irrelevant web-page section may or may not exist in-between two content segments. Additionally, they will definitely not cover the whole web-page sequence.

In the contrary, two type of boundaries are defined for our segmentation problem: opening and closing ones. In this case, a segment is defined as the sub-sequence in-between an opening and a closing boundary. We will now formally define this type of segmentation, which will be referred to as *interspersed* segmentation.

In the same way as before, let the sequence S of length n be associated with an M-length array $\langle p_1, p_2, \ldots, p_M \rangle$ of *potential boundary positions*, which specify $M+1$ adjacent non-divisible elements $\langle [0, p_1), [p1, p_2), \ldots, [p_M, n] \rangle$.

[5] P_k does not take into account whether the actual number of boundaries in each k-length window also matches or not.

An *interspersed* segmentation $\ddot{\mathbf{X}}$ is defined as a sequence of paired Boolean variables $\langle \ddot{X}_1, \ddot{X}_2, \ldots, \ddot{X}_M \rangle$, such that:

$$\ddot{X}_m = \begin{cases} (1,0) & \text{if there is an opening boundary at } p_m \\ (0,1) & \text{if there is a closing boundary at } p_m \\ (1,1) & \text{if there is an opening } \textbf{and} \text{ a closing boundary at } p_m \\ (0,0) & \text{if there is } \textbf{no} \text{ boundary at } p_m \end{cases}$$

Let $\ddot{\mathbf{X}}_i^j$ denote a subsequence of $\ddot{\mathbf{X}}$ from i to j inclusive, referred to as a *window*, and let $\Sigma\ddot{\mathbf{X}}_{i,o}^j$ and $\Sigma\ddot{\mathbf{X}}_{i,c}^j$ denote the number of opening and closing boundaries in that window respectively. Finally, let $\delta_o(\ddot{\mathbf{X}}, i, k)$ be the Boolean *opening boundary presence indicator function*, indicating whether an opening boundary is present in a k-length window starting at i:

$$\delta_o(\ddot{\mathbf{X}}, i, k) = \begin{cases} 0 & \text{if } \Sigma\ddot{\mathbf{X}}_{i,o}^{i+k-1} = 0 \\ 1 & \text{otherwise.} \end{cases}$$

and let $\delta_c(\ddot{\mathbf{X}}, i, k)$ be the Boolean *closing boundary presence indicator function*, indicating whether a closing boundary is present in a k-length window starting at i:

$$\delta_c(\ddot{\mathbf{X}}, i, k) = \begin{cases} 0 & \text{if } \Sigma\ddot{\mathbf{X}}_{i,c}^{i+k-1} = 0 \\ 1 & \text{otherwise.} \end{cases}$$

Let $\ddot{\mathbf{R}}$ and $\ddot{\mathbf{H}}$ be a reference and a hypothesized segmentation respectively. We define the segmentation evaluation metric $\ddot{\mathrm{P}}_\mathrm{k}(\ddot{\mathbf{R}}, \ddot{\mathbf{H}})$ as:

$$\ddot{\mathrm{P}}_\mathrm{k}(\ddot{\mathbf{R}}, \ddot{\mathbf{H}}) = \frac{\sum_{i=1}^{M-k+1} \mathbf{1}\Big[|\delta_o(\ddot{\mathbf{R}}, i, k) - \delta_o(\ddot{\mathbf{H}}, i, k)| + |\delta_c(\ddot{\mathbf{R}}, i, k) - \delta_c(\ddot{\mathbf{H}}, i, k)|\Big]}{M - k + 1} \tag{4.3}$$

This time, the $\ddot{\mathrm{P}}_k(\ddot{\mathbf{R}}, \ddot{\mathbf{H}})$ metric calculates the proportion of k-length windows, where $\ddot{\mathbf{R}}$ and $\ddot{\mathbf{H}}$ disagree about the existence of either opening or closing boundaries. $\ddot{\mathrm{P}}_k$ takes values between 0 and 1 similarly to the simple P_k.

In the same exact way as before we define the metric $\ddot{k}$-κ which accounts for chance agreement and will be used as an objective function for our experimental evaluation:

$$\ddot{k}\text{-}\kappa(\ddot{\mathbf{R}},\ddot{\mathbf{H}}) = \frac{1-\ddot{\mathrm{P}}_k(\ddot{\mathbf{R}},\ddot{\mathbf{H}})-\ddot{C}}{1-\ddot{C}} \tag{4.4}$$

where $\ddot{C}$ is the probability of having an agreement between $\ddot{\mathbf{R}}$ and $\ddot{\mathbf{H}}$ by chance. $\ddot{k}$-κ takes values between -1 and 1 similarly to the simple k-κ.
In all of our experiments, k-κ was used as the *objective function*.

k-precision and k-recall

Finally, we will use two secondary metrics for assessing some of the hypotheses. These are the k-precision and k-recall measures and are the windowed variants of the typical precision and recall used throughout the machine learning literature. k-precision corresponds to the proportion of all boundary positions which we predicted, that were actual boundaries and k-recall corresponds to the proportion of all actual boundary positions which we managed to predict. Both metrics introduce a near miss tolerance (as all window-based metrics do) and their values range from 0 to 1.

4.3 Experiments and Results

This section provides detailed information about the hypotheses that were set for evaluating our system, how they were tested, the results that were obtained and how these can be interpreted with respect to what was expected. Where possible, significance testing is used to further validate the outcome of the experiments.

In order to optimize the systems under testing, based on our objective function, we used the principle machine learning procedure of splitting our dataset into a train and a test set. We optimized any parameters on the train set and tested the system's performance on the test set. In order to split the full corpus we used the following number of instances from *each* source:

- **Training set:** 1000 web-pages
- **Test set:** 500 web-pages

Throughout all experiments we will compare results against:

1. Our baseline system configuration which as described in section 4.3.1.
2. Any alternatives configurations that exist regarding the hypothesis that is tested at each stage

4.3.1 Preliminaries

Summary of System Variations

Before getting into the actual experiments discussion, a summary of the component variations that can be used as alternatives and the parameters that affect their performance at each phase of our system is given in Table 4.3.

System Phase	Major Alternatives	Important Parameters
Sequence Translation	No translation	-
	Token-based translation	Type: Simple or Extended
Repeat Extraction	Simple dot-plot	-
	Enhanced Suffix Array	minlen minrep
Repeat Density Analysis	-	-
Segment Extraction	Naive	-
	Clustering	-

Table 4.3: *Summary of system phases and their alternatives/parameters*

All sets of competing alternatives and parameters will be evaluated in the following section.

Baseline

Without a doubt, the most crucial component of the system, in terms of novelty and importance is the *repeat extraction component*. It represents the center-piece around which any other component was built, in order to complement its utility. As described in detail in section 3.3.2, the repeat extraction phase includes two major alternatives, as to how repeat information is extracted: the *simple dot-plot component* and the *enhanced suffix array component*.

The dot-plot, being a far less sophisticated repeat analysis tool than enhanced suffix arrays, was chosen as the core of the baseline configuration against which any other aspect of our system will be tasted.

More specifically, a *"stripped-down"* version of the full system configuration will be considered the baseline for all performed experiments. This configuration will use:

- No sequence translation (character-based sequence representation).
- The simple dot-plot component for repeat extraction.
- The naive method for segment extraction.

Significance Testing

Where appropriate – and possible – our experiments will be accompanied by a significance test that aims at validating our results in a statistically solid manner. There are numerous types of significance tests, each suitable for various types of tasks and requiring different assumptions to be held. All tests aim to compute the probability – known as *p-value* – that the generated observations would occur by chance in a given *null* hypothesis. If this *p-value* is smallest than a given threshold (usually 0.1) then the *null* hypothesis is disproved, meaning that the observations are statistically significant.

In our context the observations would be pairs of $\ddot{k}$-κ values (one value from each system configuration being compared). The significance test applied is the one-tailed, paired t-test.

The unit for which each observation pair will be sampled is an important choice. In our initial tests, we used single web-pages as a unit for significance testing, i.e. we got a few thousands observations ($\ddot{k}$-κ value pairs) for each tested hypothesis. We observed that, even when the performance difference between the compared configurations was not too vast, the resulting *p-value* was extremely small (down to a magnitude of 10^{-7}). We concluded that *per-web-page* significance testing was not a concrete enough choice in order to validate our experiments. This is probably due to the violation of the independence assumption that t-test (as most of significance tests) requires to be held, as well as due to the huge number of observations.

As a far more conservative alternative, we will apply t-tests using web-sources as observation units. This means that for each hypothesis we will only have a sample of 7 observations. This way the *null* hypothesis will be much harder to disprove, but its significance will be much more statistically solid.

4.3.2 Evaluating the Repeat Extraction Component

In this section we will assess the performance of the *repeat extraction component* alone. The evaluation of the repeat extraction component constitutes of 2 main experiments. First, the performance of our system, after being optimized using an equal number training instances from all sources, will be evaluated. Then, we evaluate its performance, after performing separate training procedures for each source in order to assess the quality of its segmentations *in-domain*.

Our hypothesis is that the enhanced suffix array component will outperform the simple dot-plot component, both when trained on *all sources*, and when trained and tested *in-domain*. It is expected that *in-domain* performance will be superior of the generalized one.

4.3.2.1 Generalization Performance (Trained on All Sources)

For this experiment we will use a *stripped-down configuration*[6] of the system based on the enhanced suffix array component and compare it to our baseline (for both, see section 3.3.2).

In order to optimize the enhanced suffix array component's two parameters – $minlen$ and $minrep$ –, we ran a full parameter sweep[7], using instances from **all** sources. The pair of optimized parameters that resulted in the best mean $\ddot{k}$-κ value on the training set was:

- $minlen = 25$
- $minrep = 9$

These parameter values were then used to test the generalization performance of the system configuration on the test set. Table 4.4 shows the results in terms of average $\ddot{k}$-κ over **all** test instances.

	$\ddot{k}$-κ
Baseline	0.02017
En. Suf. Array *(No translation)*	0.11164

Table 4.4: *Generalization performance of stripped-down configuration based on en. suf. array (trained on all sources)*

[6] i.e. no translation and no segment clustering.

[7] $minlen : \{5, 15, 25, 35, 45\}$, $minrep : \{3, 5, 7, 9\}$. See Appendix A for parameter sweep plots.

We observe that even the most stripped-down version of our system based on the enhance suffix array component outperforms the baseline (dot-plot). More specifically the baseline's performance is only slightly above random [8]. While our the enhanced suffix array configuration does not perform much superiorly than random, it is a definite improvement.

In order to get a more clear idea regarding where both our system configurations perform well – and where not –, we averaged the $\ddot{k}$-κ output of the experiment per source Our expectation is that the performance will vary a lot from one source to another. Fig. 4.2 illustrates the results.

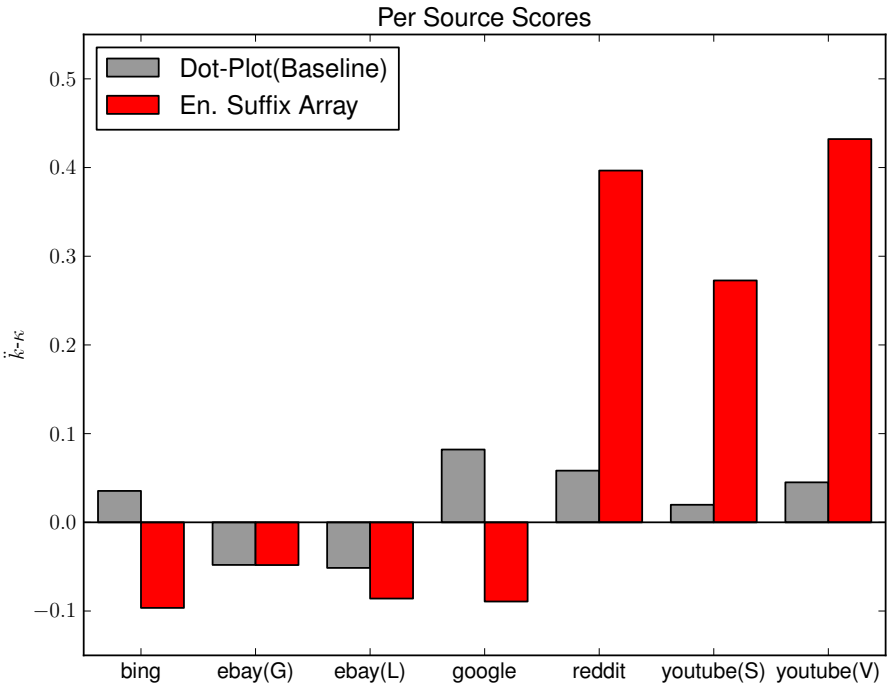


Figure 4.2: $\ddot{k}$-κ *values per source after training using all sources*

Our expectation is confirmed. Firstly, the baseline configuration is steadily performing close to random (its best output is slightly below 0.1, for Google). In the contrary, the enhance suffix array configuration exhibits highly varied results for each source. Its performance ranges from worse than random, to the surprisingly good results for the *Reddit* and *Youtube Videos* sources. The huge structural dissimilarities between sources is, without a doubt, a deciding factor for this outcome.

[8] Recall that a $\ddot{k}$-κ value of 0 indicates chance agreement.

Furthermore, we expect that the fact that the baseline does outperform the enhanced suffix array configuration in some of the sources a result of placing potential segments throughout the web-page sequences. In order to assess this claim, we calculated k-precision and k-recall values for all test instances. The resulting mean values are shown in Table 4.5.

	k-**precision**	k-**recall**
Baseline	0.27244	0.7735
En. Suf. Array *(No Translation)*	0.30334	0.54492

Table 4.5: *k-precision and k-recall values*

Once again the results confirm our expectations. The combination of a very high k-recall and a contrastingly low k-precision, resulting from the baseline's output, is indicative of the fact that the dot-plot component tends to extract segments all over the web-page's sequence. While it may end up estimating the true segments, it also outputs numerous non-existent ones.

In order to validate the statistical significance of these results, we performed a one-tailed, paired t-test using 7 $\ddot{k}$-κ value pairs – one for each source – as these were illustrated in the bar chart of Fig. 4.2. The result of the t-test is shown in Table 4.6.

Comparison			p-**value**
Baseline	**vs.**	En. Suf Array *(No Translation)*	0.1667

Table 4.6: *T-test significance testing for the basic system configuration*

Although we did not get a p-value clearly indicating statistical significance[9], the resulting value is promising enough, considering that we used the least sophisticated system configuration based on enhanced suffix arrays.

[9] A p-value < 0.1 is required for this.

4.3.2.2 Generalization Performance (Trained on each Source Separately)

Probably the most clear conclusion drawn from the previous experiment was the inability of the particular system configuration to generalize when optimized throughout all sources.

In this experiment, our goal is to investigate whether the system's performance would be improved, when trained specifically for a single source. Although this will definitely not constitute concrete evidence for its robustness, it will be an important indication of its segmentation capabilities, given the proper set of parameters.

In order to perform the experiment we trained the system a total of 7 times – once for each source – resulting in 7 *minlen-minrep* parameter pairs[10]. The optimized parameters for each source are shown in Table 4.7.

Source	*minlen*	*minrep*
Bing	45	5
Ebay (List)	15	9
Ebay (Grid)	5	3
Google	45	9
Reddit	25	3
Youtube (Results)	45	7
Youtube (User V.)	35	5

Table 4.7: *Separately optimized parameters for each source*

Subsequently, using the respective parameter pair for each source, we tested the system's performance by running 7 separate experiments, which resulted in 7 separate $\ddot{k}$-κ performance values. Table 4.8 shows the mean of these 7 values.

	Mean $\ddot{k}$-κ
Baseline	0.02017
En. Suf. Array *(No Translation)*	0.22537

Table 4.8: *Mean $\ddot{k}$-κ value, averaged over the 7 system runs, each optimized specifically for a single source*

[10]Once again with no translation/no segment clustering

As expected, the mean performance across all 7 experiments is significantly higher compared to the previous experiment. A more detailed illustration (Fig. 4.3) shows the $\ddot{k}$-κ value for each source, once again compared to the previously obtained values from our baseline. This time, all $\ddot{k}$-κ values for the enhanced suffix array component are positive (indicating segmentation agreement with the reference) and better than the respective baseline performance (except from Bing's score).

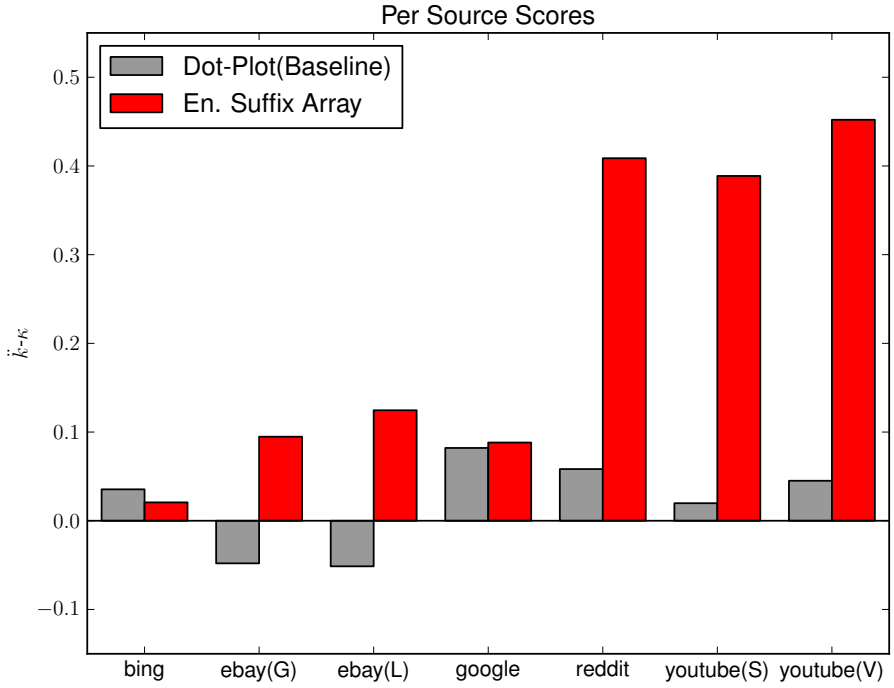


Figure 4.3: $\ddot{k}$-κ *value of 7 in-domain experiments, each optimized specifically for a single source*

Finally, the mean *k*-precision and *k*-recall values across the 7 experiments are shown in Table 4.9, once again indicating significant improvement compared to the previous experiment.

	***k*-precision**	***k*-recall**
Baseline	0.27244	0.7735
En. Suf. Array *(No Translation)*	0.37689	0.62595

Table 4.9: *k-precision and k-recall values*

At this point, it should be noted, that the results of this experiments should be approached with caution. Definitely, they present promising evidence that, given the appropriate parameters, this *basic* system configuration can perform fairly well *in-domain*, regardless not using some of the more sophisticated component variations that are available. However, this experiment provides no evidence of generalization capabilities of the system. Each system run was optimized for the particular source it was tested for. Further experiments, which will focus on optimizing parameters over all sources (as in the first experiment) will provide more concrete evidence on the system's capabilities.

4.3.3 Evaluating the Effect of Sequence Translation

Up to this point, the experiments that have been presented, only focused on the repeat extraction component, without accounting for other components' variations. In fact, the configurations used in those experiments have not utilized any of the more sophisticated alternatives that the *sequence translation* and the *segment extraction* components introduce.

In this experiment we will evaluate the *sequence translation* component's effect on the overall system performance. Recall the tree alternative translation policies that were introduced in section 3.3.1:

- No translation *(character-based sequence)*
- Simple translation *(simple token-based sequence)*
- Extended translation *(extended token-based translation)*

Our hypothesis is that each translation configuration will significantly affect the overall performance of the system, both in terms of segmentation results, as well as regarding the repeat extraction parameters that will be used to obtain those results. We want to investigate these effects.

Character-based sequences have already been tested and compared to our baseline in the first two experiments presented. For this experiment we separately trained the system for the two token-based translations as well[11]. This way, we obtained two more *minlen-minrep* parameter pairs, which are shown in Table 4.10.

[11] On the full training set.

Translation Policy	*minlen*	*minrep*
None	25	9
Simple	3	7
Extended	9	9

Table 4.10: *Optimized parameters for each sequence translation policy*

Before moving on to the experimental results, the optimized parameters for each translation policy already provide an interesting observation. The optimal value for the *minlen* parameter significantly differs from one policy to another. In particular, the character-based sequences are better suited for high *minlen* values (25), followed by the extended token-based sequences (9) and, lastly, by the simple token-based ones (3). This makes intuitive sense, since, in general, character-based sequences are obviously far longer compared to extended token-based ones, which are, again, longer than simple token-based sequences. More on this subject will be discussed in 4.3.4.

Using these three parameter settings for the enhanced suffix array component, each combined with its respective sequence representation, we tested their generalization performance by running the system on our full test set. Table 4.11 shows the resulting $\ddot{k}$-κ values obtained from each run.

	Translation	$\ddot{k}$-κ
Baseline	None	0.02017
En. Suf. Array	None	0.11164
	Simple	0.17286
	Extended	0.17476

Table 4.11: *Generalization performance of each translation policy*

The resulting performance measurements indicate that token-based translation policies do in fact improve the performance of the system. Both perform better than the baseline and the ”no-translation” system configurations used in previous experiments.

Recalling the reasons that led to the design of these translation policies, token-based sequences can definitely reduce the amount of *noise* that is a characteristic of character-based ones. They provide a level of sequential abstraction that is not affected by insignificant character mismatches, while still accounting for the overall structure of the web-page in terms of html-tags, tag parameters and number of content words.

In terms of their performance, the two token-based policies do not differ significantly in order to extract safe conclusion regarding which one should be preferred.

A summary of the generalization performance of each policy per data source is given in Fig. 4.4.

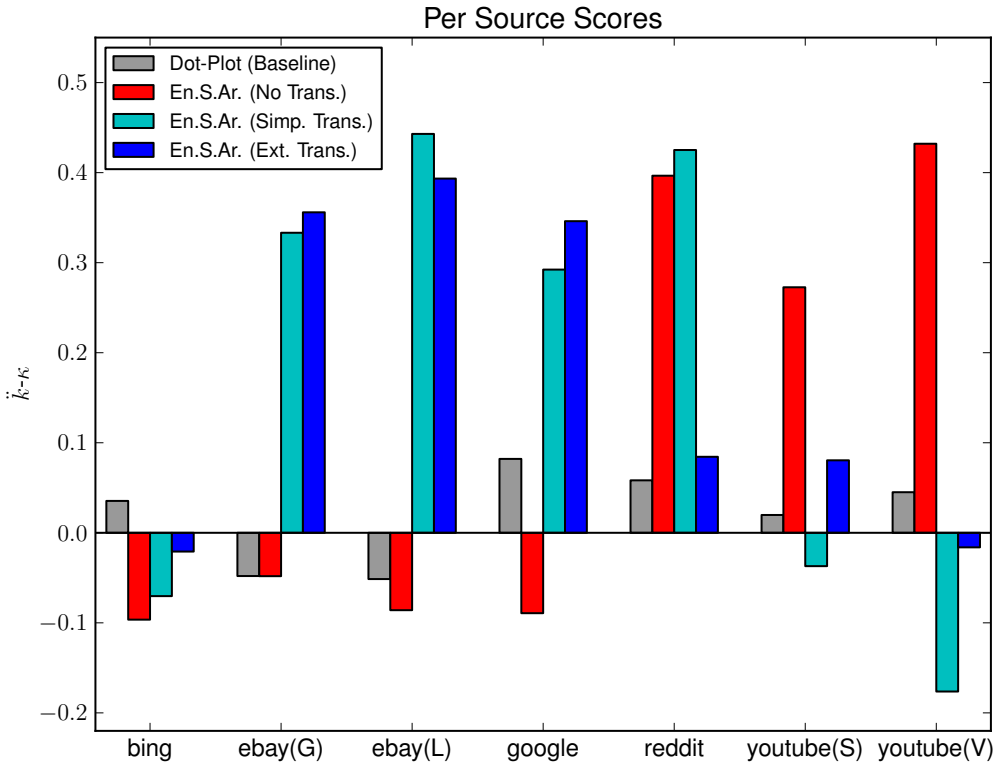


Figure 4.4: *Generalization performance of translation policies per source*

It is evident that different policies are better suited for different sources. In general, token-based ones tend to perform better, although interestingly enough, not translated sequences perform very well on the two Youtube sources, as well as on Reddit. Finally, the Bing web-pages seem to be rather problematic since they result in negative $\ddot{k}$-κ values for all translation settings. A possible explanation is that Bing tends to include numerous irrelevant elements in-between its results, that may negatively influence the segmentation performance. Still, more detailed investigation, regarding what makes each source better suited for a particular translation policy, should be performed, but is outside the scope of this project.

Finally, significance testing was performed in order to validate whether the performance improvement that token-based translation policies resulted in were statistically concrete. Once again, we used 7 pairs of averaged $\ddot{k}$-κ values – one for each source – in order to compare the two policies to our baseline. Table 4.12 shows the resulting p-values.

***t*-test**

Comparison			p**-value**
Baseline	**vs.**	En. Suf. Array *(Simple Translation)*	0.09925
Baseline	**vs.**	En. Suf. Array *(Extended Translation)*	0.05224

Table 4.12: *T-test significance testing for both translation policies*

This time, both system configurations under testing performed well enough in order for the results to be statistically significant. The *null* hypothesis, that their performance could have resulted by chance is disproved.

4.3.4 Effect of *minlen* and *minrep* parameters

In this section a brief discussion on how the two main parameters of the enhanced suffix array repeat extraction component affect the performance of the system is provided.

The most obvious conclusion that was extracted from the parameter sweeps performed at the training stage of our experiments is that the *minrep* parameter has minimum effect on the overall system performance[12]. For any tested value, the performance changes were minimal both per source and in the full corpus.

On the other hand *minlen*'s effect on the quality of segmentation is far stronger. As Fig. 4.5 shows, *minlen* is a deciding factor that heavily influences performance on most sources (dotted lines) and on the full corpus (thick line)[13].

[12] See Appendix A for all diagram tables of the parameter sweeps.
[13] A *minrep* value of 7 was used for these illustrations.

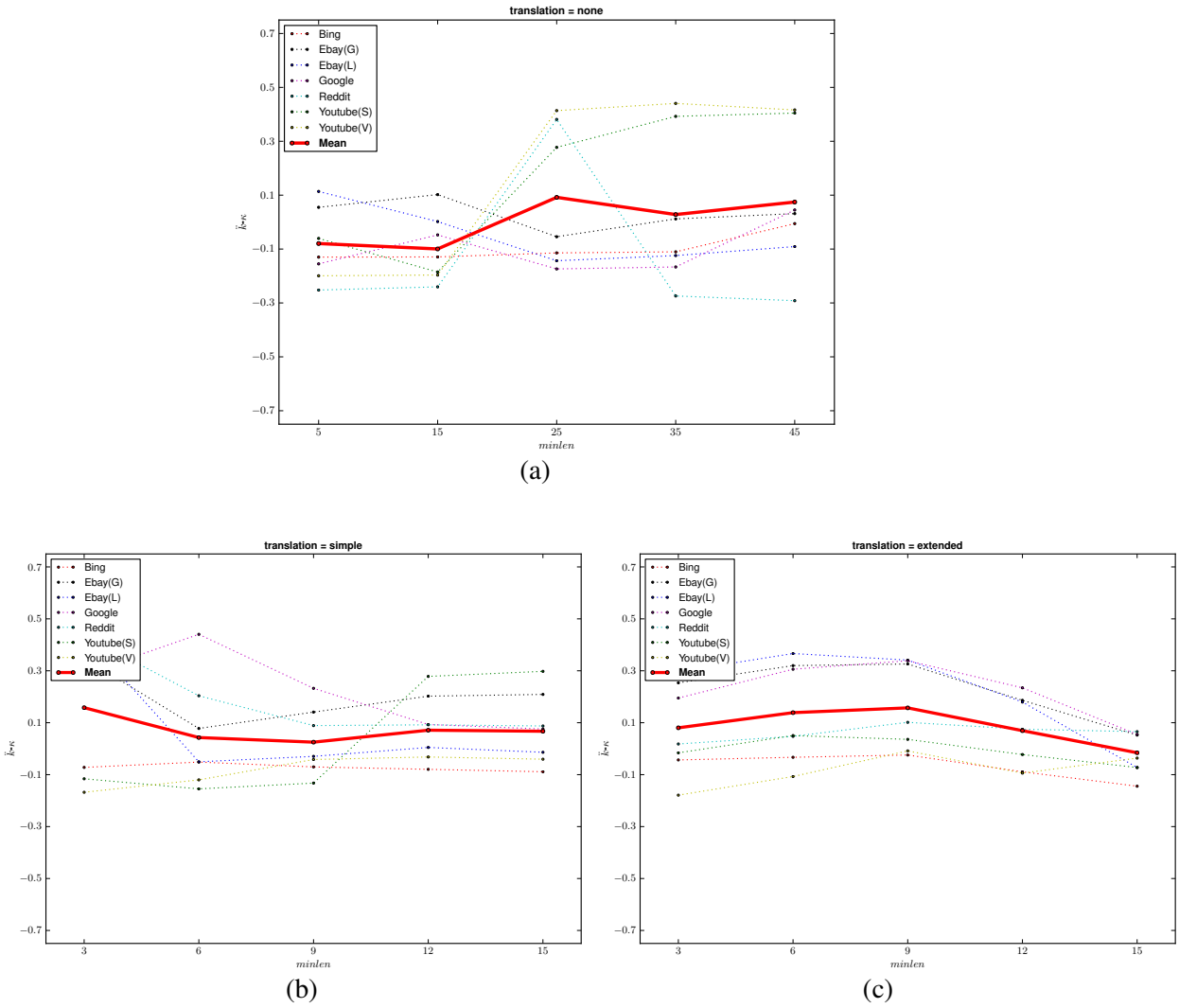


Figure 4.5: *Segmentation performance effect of minlen parameter in (a) character- and (b,c) token-based sequences*

In particular, in some cases, the performance shifts from one value of *minlen* to another can be rather erratic. The Reddit and both Youtube sources – for the case of "no translation" policy – are good examples. All three sources exhibit a big increase in performance at the $minlen = 25$ mark, while the rest of the sources appear to be less influenced. Reddit has the most erratic behaviour, since it immediately falls back to worse than random performance.

The most interesting observation that can be extracted from these figures is the *smoothing* effect that token-based translation has to this erratic behaviour. It is easily observed that plots (b) and (c), corresponding to token-based translation, appear to

have fewer shifts in performance for various values of $minlen$. This effect is even more notable in the case of the extended token-based translation, where the performance on all sources follows a smooth curve that has a maximum for $minlen = 9$.

This observation is supporting evidence of the positive effect that sequence translation can have to the overall behaviour of the system.

4.3.5 Effect of Clustering on Segment Extraction

The last factor of the system that will be evaluated is the segment extraction phase. Recalling what was described in 3.3.4, the segment extraction component includes two main alternatives; the naive extraction method and the clustering aided extraction method.

The naive approach makes no distinction between the regions of high repeat density (peaks), extracting segments from all of them. The clustering method tries to divide the peaks into coherent groups and then, select the group that appears to be more dominant, in terms of the area it covers on the web-page.

Our hypothesis is that clustering should result in a significant improvement in segmentation performance. It is inevitable that regions of high repeat density will appear even in parts of the web-page sequence that do not correspond to content sections. Clustering appears to be a simple, yet effective way of dealing with this issue.

In order to assess this hypothesis, all optimized system configurations tested in the previous experiments[14] are now tested by enabling the segment clustering component.

Table 4.13 shows the results of this experiment by comparing previously reported performance measures (no clustering) with the newly obtained ones (with clustering).

[14] See Table 4.10.

	Translation	Segment Extraction	Mean $\ddot{k}$-κ	k-precision	k-recall
Baseline	None	w/o clustering	0.02017	0.27244	0.77350
En. Suf. Array	None	w/o clustering	0.11164	0.30334	**0.54492**
		with clustering	**0.27459**	**0.44238**	0.53904
	Simple	w/o clustering	0.17286	0.33834	**0.63968**
		with clustering	**0.32622**	**0.46716**	0.59947
	Extended	w/o clustering	0.17476	0.34886	**0.63520**
		with clustering	**0.23773**	**0.39774**	0.58269

Table 4.13: *Generalization performance comparison with and without segment clustering*

The difference in performance for all three enhanced suffix array-based configurations is impressively vast. Clustering seems to provide the biggest performance *boost* of all aforementioned system variations. This is evident both in terms of $\ddot{k}$-κ and k-precision. k-recall values have slightly diminished, but the difference is insignificant and probably expected, since proper segments are going to be discarded by the clustering algorithm from time to time.

As a final validation of these results we performed two groups of significance tests. The first one (Table 4.14) is between our baseline and all three clustering-aided system configurations.

***t*-test**

Comparison			p-value
Baseline	**vs.**	En. Suf. Array *(No Translation)* *(Segment Clusters)*	0.03713
Baseline	**vs.**	En. Suf. Array *(Simple Translation)* *(Segment Clusters)*	0.01672
Baseline	**vs.**	En. Suf. Array *(Extended Translation)* *(Segment Clusters)*	0.00885

Table 4.14: *T-tests between clustering configurations and our baseline*

The second, and far more interesting one is between the clustering aided system configurations and their naive counterparts (Table 4.15).

***t*-test**

Comparison			***p*-value**
En. Suf. Array *(No Translation)* *(Segments: Naive)*	**vs.**	En. Suf. Array *(No Translation)* *(Segments: Clusters)*	0.00197
En. Suf. Array *(Simple Translation)* *(Segments: Naive)*	**vs.**	En. Suf. Array *(Simple Translation)* *(Segments: Clusters)*	0.00040
En. Suf. Array *(Extended Translation)* *(Segments: Naive)*	**vs.**	En. Suf. Array *(Extended Translation)* *(Segments: Clusters)*	0.08006

Table 4.15: *T-tests between clustering configurations and their naive alternatives*

All 6 performed t-tests resulted in p-values < 0.1, providing statistical significance to our results. Adding segment clustering to the optimized enhanced suffix array configurations, improved their performance in an impressive way, providing a final validation to our novel system's overall segmentation capabilities.

Chapter 5

Conclusions

For this project, we designed, built and thoroughly tested a novel web-page segmentation system to be used in the context of Information Retrieval content extraction. The novelty aspect of our approach lies at the utilization of repeat analysis techniques, in combination with the sequential view of web-pages, in order to identify repeated patterns in web-pages, that are expected to correspond to significant content sections.

The system that was built, consisted of four major components working together, each one devoted to an independent phase of the web-page segmentation algorithm. Our main goal was to assess the suitability of each component and to critically compare the possible variations and parameters that could be used at each operational phase.

In order to do so we set a number of hypotheses to be experimentally tested and evaluated. Each hypothesis that was designed, focused on a different factor that may or may not affect the systems performance significantly. These hypothesis can be briefly summarized as follows:

- Investigate the suitability of repeat extraction and analysis as the basic tool for identifying content sections of web-pages. In particular the two components – a baseline, based on dot-plots and a sophisticated enhanced suffix array – were tested and compared. Our expectation was that enhanced suffix arrays should be far more powerful and flexible than the baseline.
- Assess the flexibility that the sequential view of web-pages provides to the particular task. Test and compare each of the three levels of abstraction that were designed for this project.
- Investigate how the segment extraction phase of the system can be improved by using a novel clustering-based method for distinguishing between relevant and irrelevant regions of high repeat density.

By performing thorough experiments we came up with a set of impressive results regarding those hypotheses. Most of our expectations were met and in some cases the outcomes of the experiments were even more promising than expected.

Summarizing these outcomes we can conclude that:

- The enhanced suffix array is a powerful tool that can be used as the basis of a web-page segmentation system based on repeat analysis. It can adapt to various types of web-page sources and perform particularly well in all of them, given the appropriate parameters.

- Even more importantly, when combined with an appropriate type of sequential abstraction, the resulting repeat density analysis can result in a better and more easily generalizeable system.

- On top of these main phases of the system's pipeline, the novel clustering approach for the extraction of the final segments resulted in surprisingly impressive segmentation quality. The concrete statistical evidence – coming from significance testing – further solidify the fact that segment clustering should be an extremely important factor for further extensions and variations of this approach.

Despite the overwhelmingly promising results, some concerning factors were identified. Most importantly, the highly diverse nature of web-pages makes the generalization performance of our method suffer in cases of uniquely structures web-domains. Although using sequential representation abstraction techniques, indicated that smoother generalization is possible, such approaches need far more testing and improvements to be considered usable in a greater scale.

Additionally, the discrete parameter space of this approach makes its optimization procedure a non-trivial task. Although intense experiments were performed, the total search space was definitely not explored thoroughly enough in the scope of this project.

Finally, the adaptable nature of our approach has already given answers to some of the limitations of existing web-page segmentation efforts, even at this early stage. Most importantly, since we are focused on the context of content extraction, its ability to extract content sections that correspond to the most dominant segments of the page, while ignoring the rest – thanks to the segment clustering component – constitutes a major improvement compared to existing techniques.

5.1 Future Work

The promising results that were obtained from this novel project should ideally be the starting point for a much more detailed research investigation of the utility of repeat analysis techniques for the task of web-page segmentation. Some of the main focuses of further work on this approach should include:

- Much more detailed investigation of the effect of sequence translation should be performed. This is significantly important, since evidence showed that there is much room for improvement in terms of the generalization capabilities of our system.
- As in every machine learning task, the collection of even more diverse data for optimizing and evaluating the system's performance should definitely be beneficiary.
- More principled ways of optimizing the system's components – especially the repeat extraction phase – should be investigated. Parameter sweeps, although useful, have limited capabilities at exploring the full parameter search space.
- The system was able of loosely identifying the regions that corresponded to content segments. However, much more sophisticated techniques can be applied in combination with the very effective segment clustering method that is already used. Sequence alignment techniques, that could make sure that the extracted segments consist of roughly the same sequences of html-tags should be a promising way to further improve the overall performance of the system.

Bibliography

Abouelhoda, M. and Ghanem, M. (2010). String mining in bioinformatics. *Scientific Data Mining and Knowledge Discovery*, pages 207–247.

Abouelhoda, M. I., Kurtz, S., and Ohlebusch, E. (2002). The enhanced suffix array and its applications to genome analysis. *Algorithms in Bioinformatics*, pages 449–463.

Abouelhoda, M. I., Kurtz, S., and Ohlebusch, E. (2004). Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86.

Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. 1215:487–499.

Beeferman, D., Berger, A., and Lafferty, J. (1999). Statistical models for text segmentation. *Machine learning*, 34(1-3):177–210.

Bernstein, M., Bolter, J. D., Joyce, M., and Mylonas, E. (1991). Architectures for volatile hypertext. *Proceedings of the third annual ACM conference on Hypertext*, pages 243–260.

Braun, J. V. and Muller, H.-G. (1998). Statistical methods for dna sequence segmentation. *Statistical Science*, pages 142–162.

Brown, N. P., Whittaker, A. J., Newell, W. R., Rawlings, C. J., and Beck, S. (1995). Identification and analysis of multigene families by comparison of exon fingerprints. *Journal of molecular biology*, 249(2):342–359.

Cai, D., Yu, S., Wen, J., and Ma, W. (2003). Extracting content structure for web pages based on visual representation. *Web Technologies and Applications*, pages 406–417.

Chakrabarti, D., Kumar, R., and Punera, K. (2008). A graph-theoretic approach to webpage segmentation. *Proceedings of the 17th international conference on World Wide Web*, pages 377–386.

Church, K. W. and Helfman, J. I. (1993). Dotplot: A program for exploring self-similarity in millions of lines of text and code. *Journal of Computational and Graphical Statistics*, 2(2):153–174.

Delcher, A. L., Kasif, S., Fleischmann, R. D., Peterson, J., White, O., and Salzberg, S. L. (1999). Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376.

Dunham, I., Hunt, A., Collins, J., Bruskiewich, R., Beare, D., Clamp, M., Smink, L., Ainscough, R., Almeida, J., Babbage, A., et al. (1999). The dna sequence of human chromosome 22. *Nature*, 402(6761):489–495.

Finn, A., Kushmerick, N., and Smyth, B. (2001). Fact or fiction: Content classification for digital libraries.

Gibbs, A. J. and Mcintyre, G. A. (1970). The diagram, a method for comparing sequences. *European Journal of Biochemistry*, 16(1):1–11.

Gusfield, D. (1997). *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press.

Helfman, J. (1996). Dotplot patterns: a literal look at pattern languages. *TAPOS*, 2(1):31–41.

Helfman, J. I. (1994). Similarity patterns in language. pages 173–175.

Kim, D. K., Sim, J. S., Park, H., and Park, K. (2003). Linear-time construction of suffix arrays. *Combinatorial Pattern Matching*, pages 186–199.

Kohlschütter, C. and Nejdl, W. (2008). A densitometric approach to web page segmentation. *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1173–1182.

Kurtz, S. (1999). Reducing the space requirement of suffix trees. *Software-Practice and Experience*, 29(13):1149–71.

Kurtz, S., Choudhuri, J. V., Ohlebusch, E., Schleiermacher, C., Stoye, J., and Giegerich, R. (2001). REPuter: the manifold applications of repeat analysis on a genomic scale. *Nucleic acids research*, 29(22):4633–4642.

lai Chung, F., chung Fu, T., Ng, V., and Luk, R. W. P. (2004). An evolutionary approach to pattern-based time series segmentation. *IEEE Transactions on Evolutionary Computation*, 8(5):471–489.

Lavrenko, V., Schmill, M., Lawrie, D., Ogilvie, P., Jensen, D., and Allan, J. (2000). Mining of concurrent text and time series. *KDD-2000 Workshop on Text Mining*, pages 37–44.

Maizel, J. V. and Lenk, R. P. (1981). Enhanced graphic matrix analysis of nucleic acid and protein sequences. *Proceedings of the National Academy of Sciences*, 78(12):7665–7669.

Manber, U. and Myers, G. (1993). Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948.

Niekrasz, J. and Moore, J. D. (2010). Unbiased discourse segmentation evaluation. *Spoken Language Technology Workshop (SLT), 2010 IEEE*, pages 43–48.

Pevzner, L. and Hearst, M. A. (2002). A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1):19–36.

Ponte, J. M. and Croft, W. B. (1997). Text segmentation by topic. *Research and Advanced Technology for Digital Libraries*, pages 113–125.

Shriberg, E., Stolcke, A., Hakkani-Tur, D., and Tur, G. (2000). Prosody-based automatic segmentation of speech into sentences and topics. *Speech Communication*, 32(1-2):127–154.

Sibson, R. (1973). Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34.

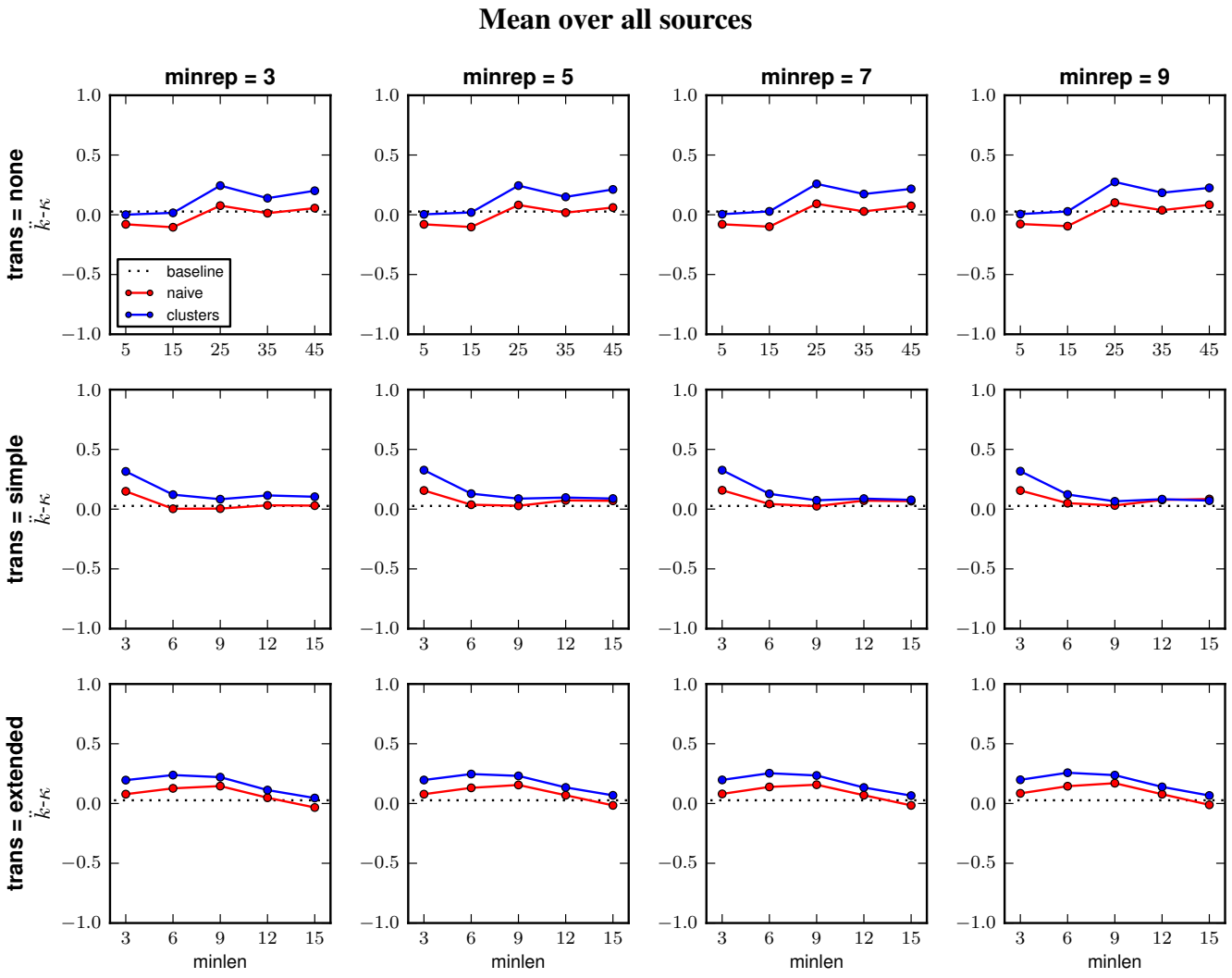
Song, R., Liu, H., Wen, J.-R., and Ma, W.-Y. (2004). Learning block importance models for web pages. *Proceedings of the 13th international conference on World Wide Web*, pages 203–211.

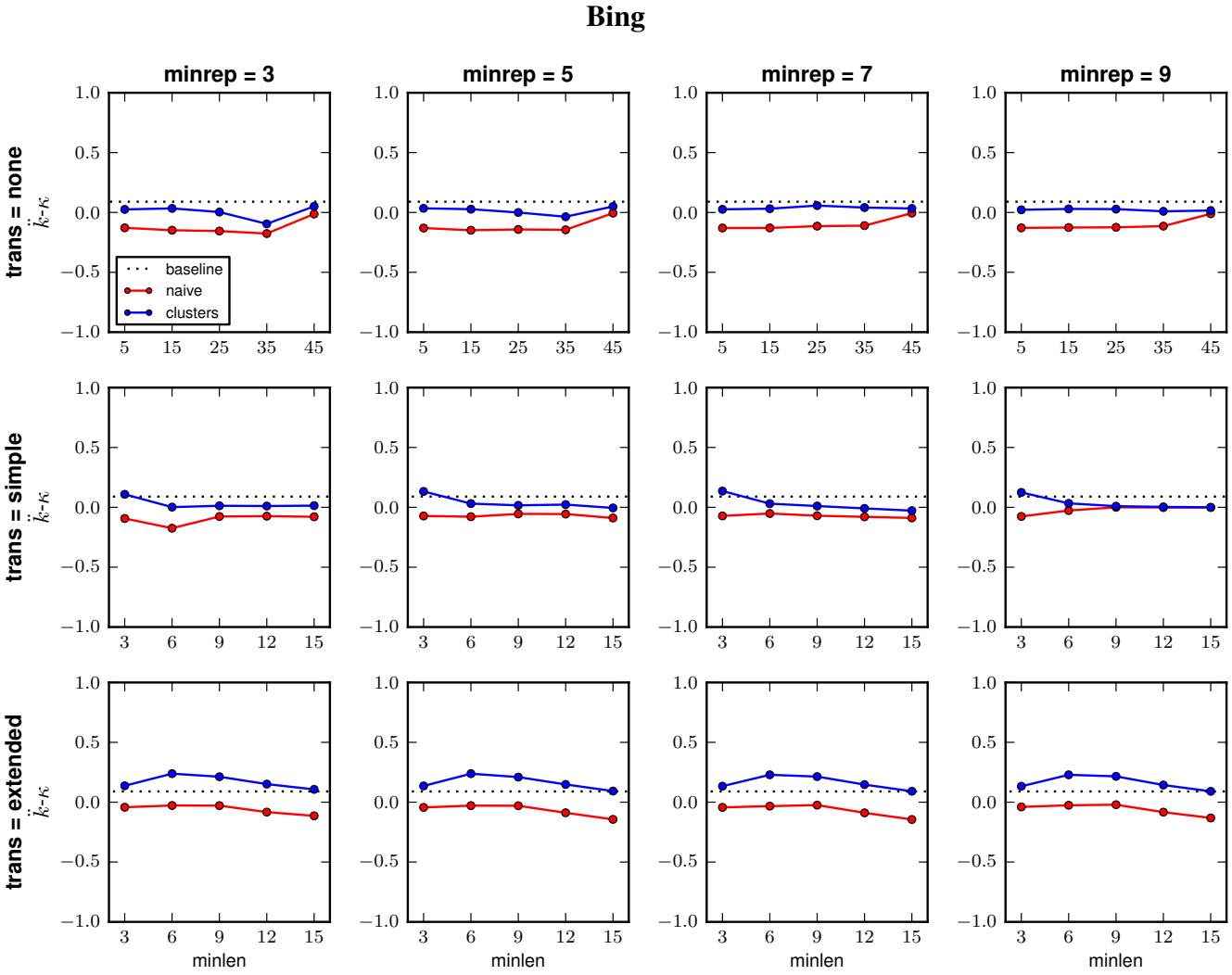
Welch, T. A. (1984). A technique for high-performance data compression. *Computer*, 17(6):8–19.

Yu, S., Cai, D., Wen, J.-R., and Ma, W.-Y. (2003). Improving pseudo-relevance feedback in web information retrieval using web page segmentation. *Proceedings of the 12th international conference on World Wide Web*, pages 11–18.

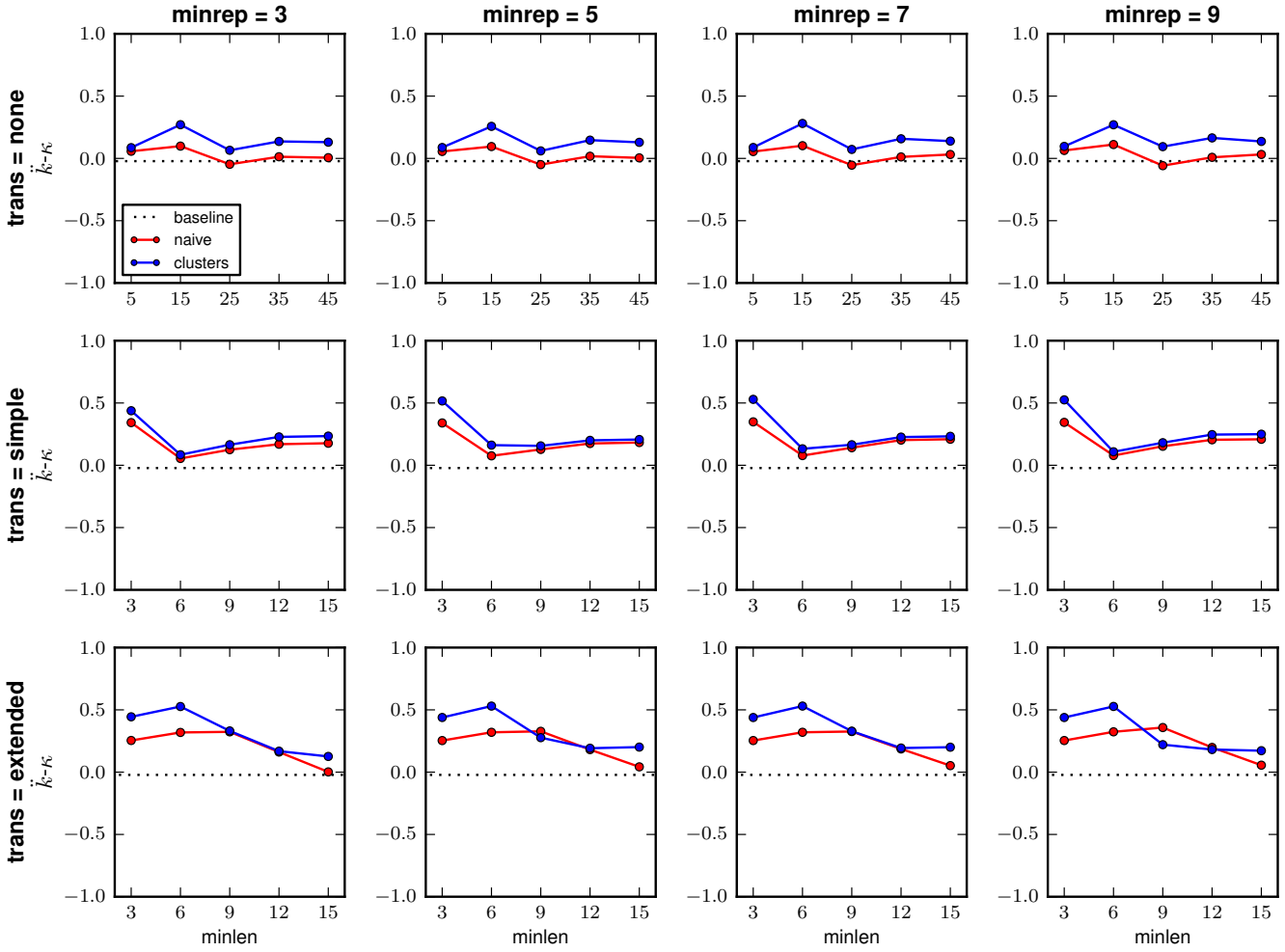
Appendix A

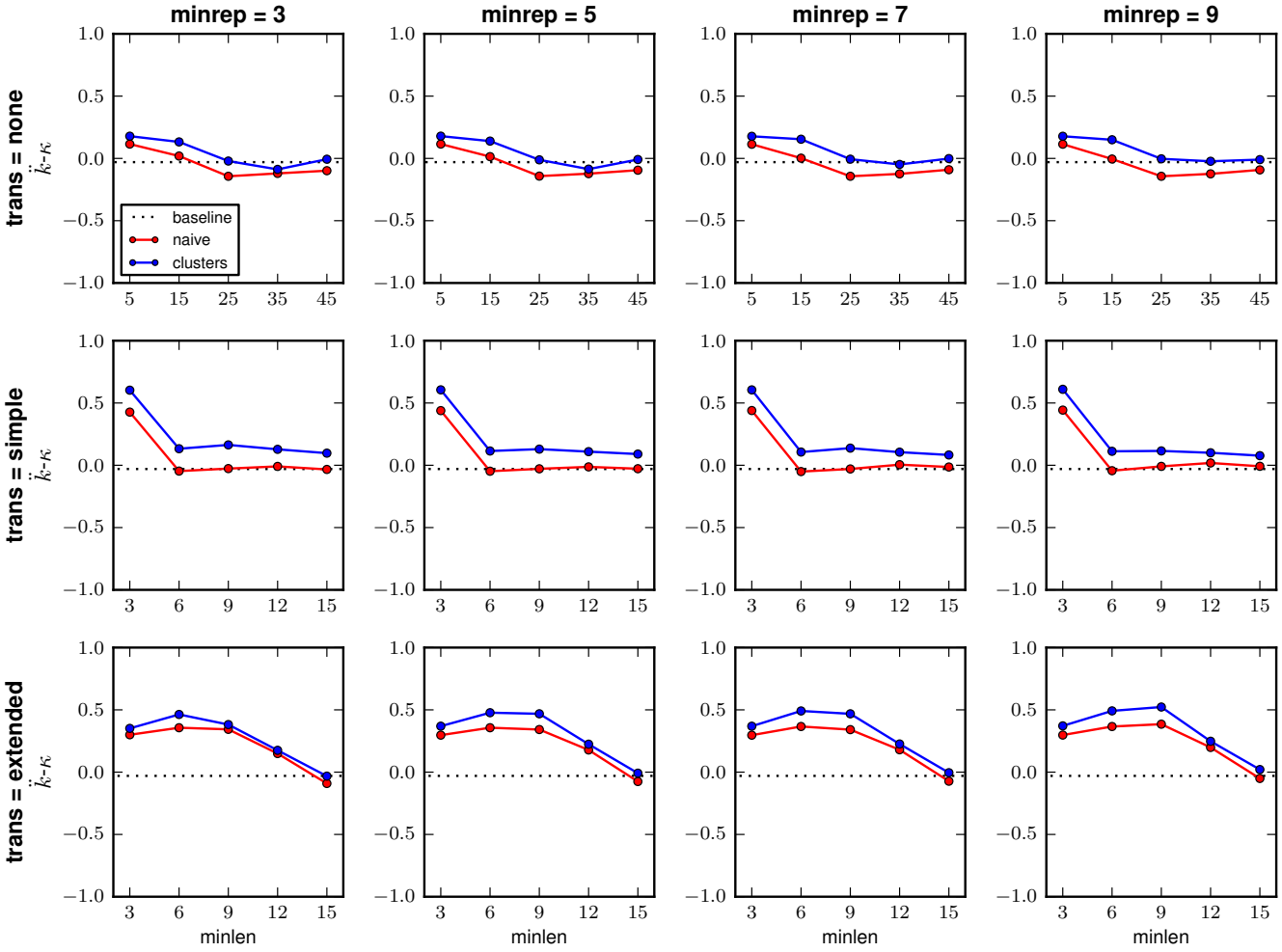
Configuration Sweeps

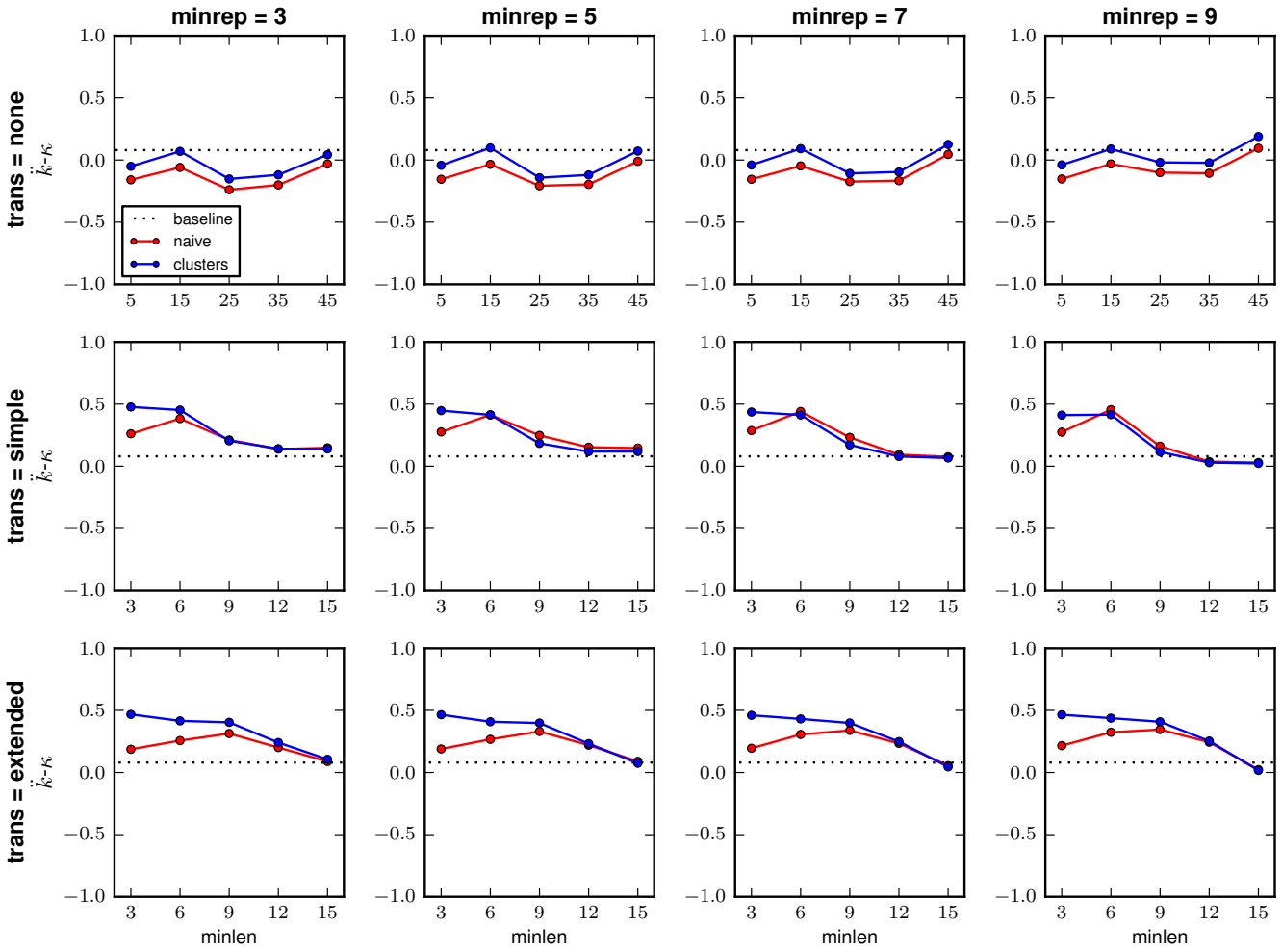


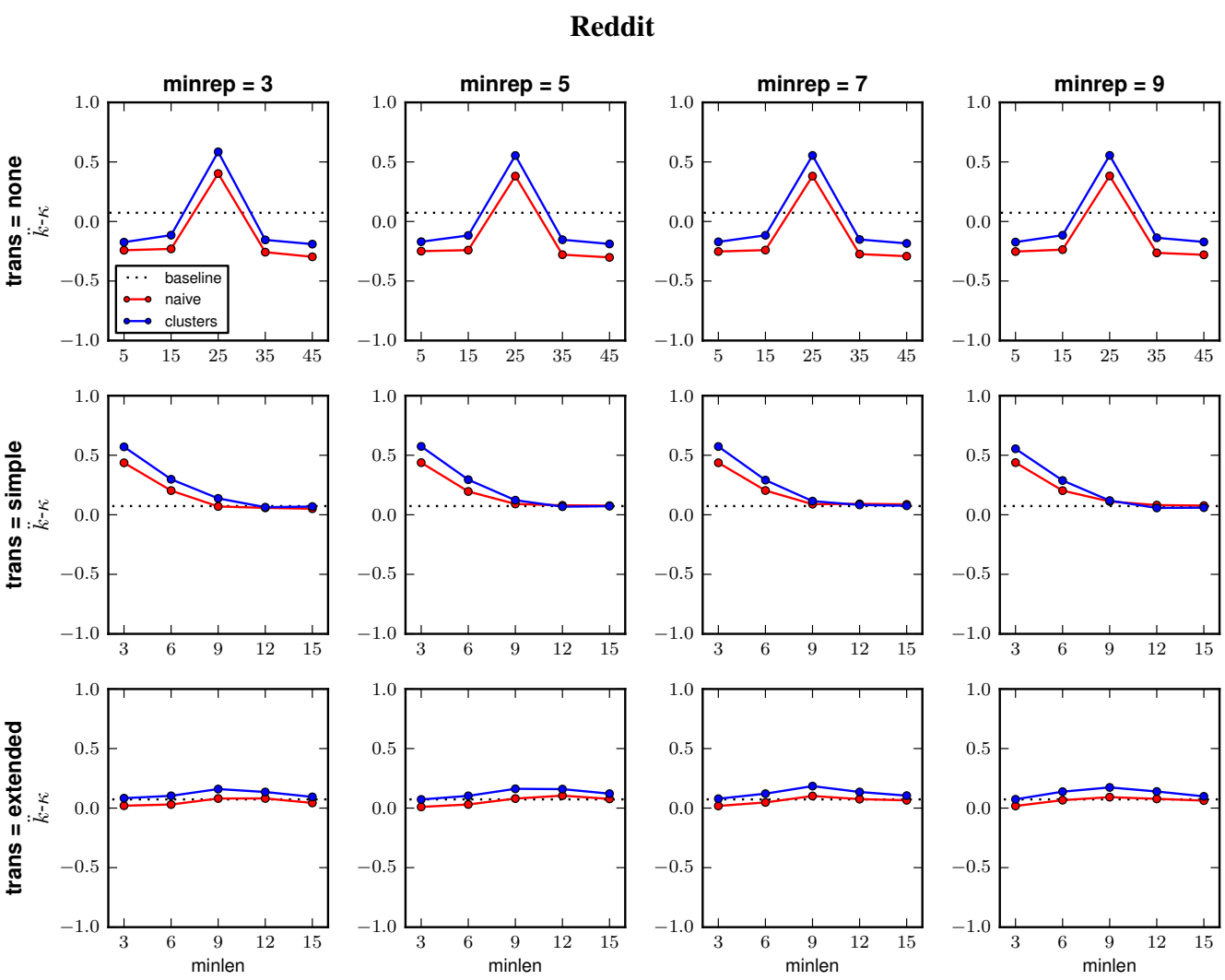

Bing
minrep = 3
minrep = 5
minrep = 7
minrep = 9
trans = none
trans = simple
trans = extended
$\ddot{k}$-κ
baseline
naive
clusters
minlen

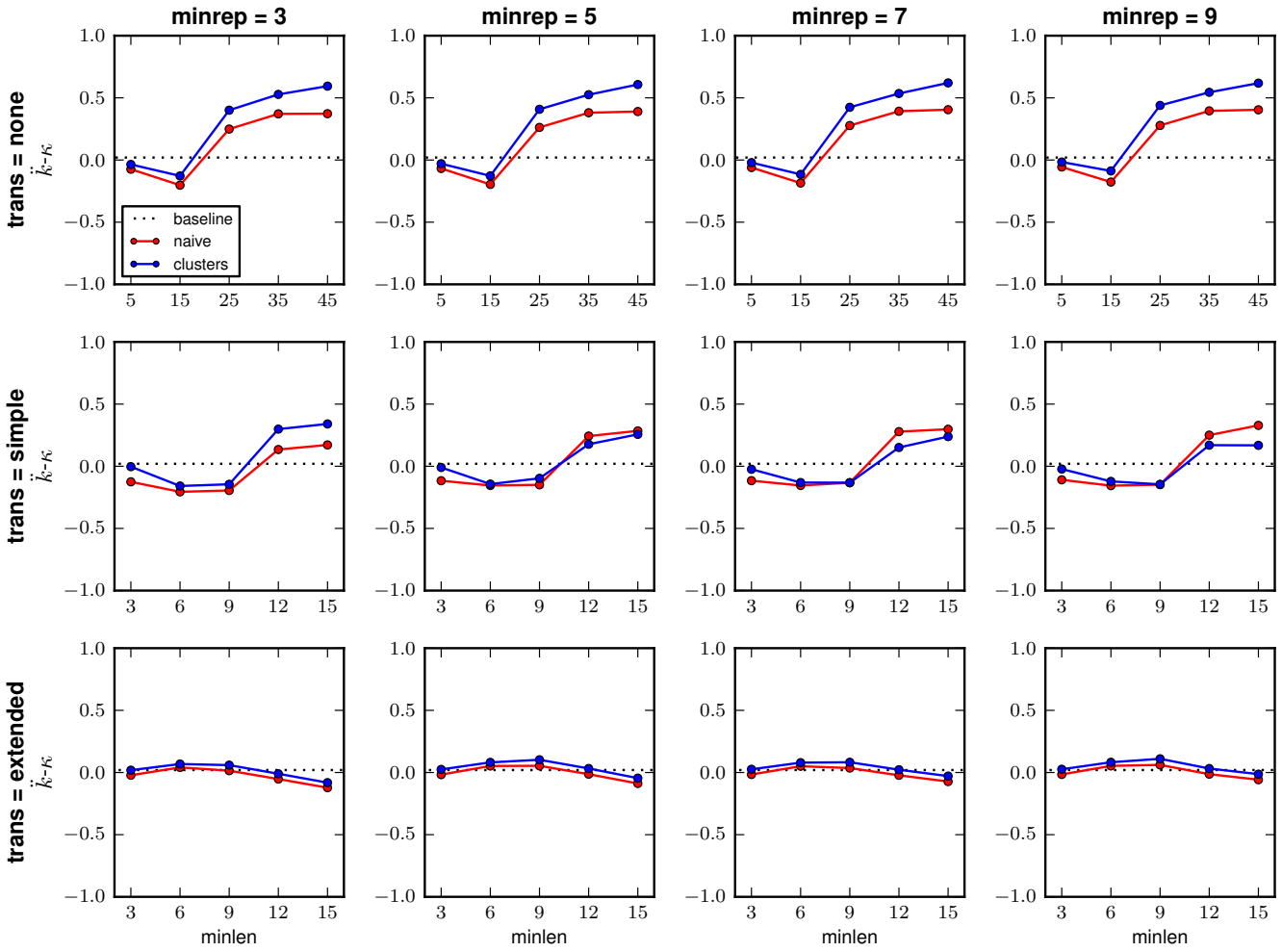
Ebay (List)

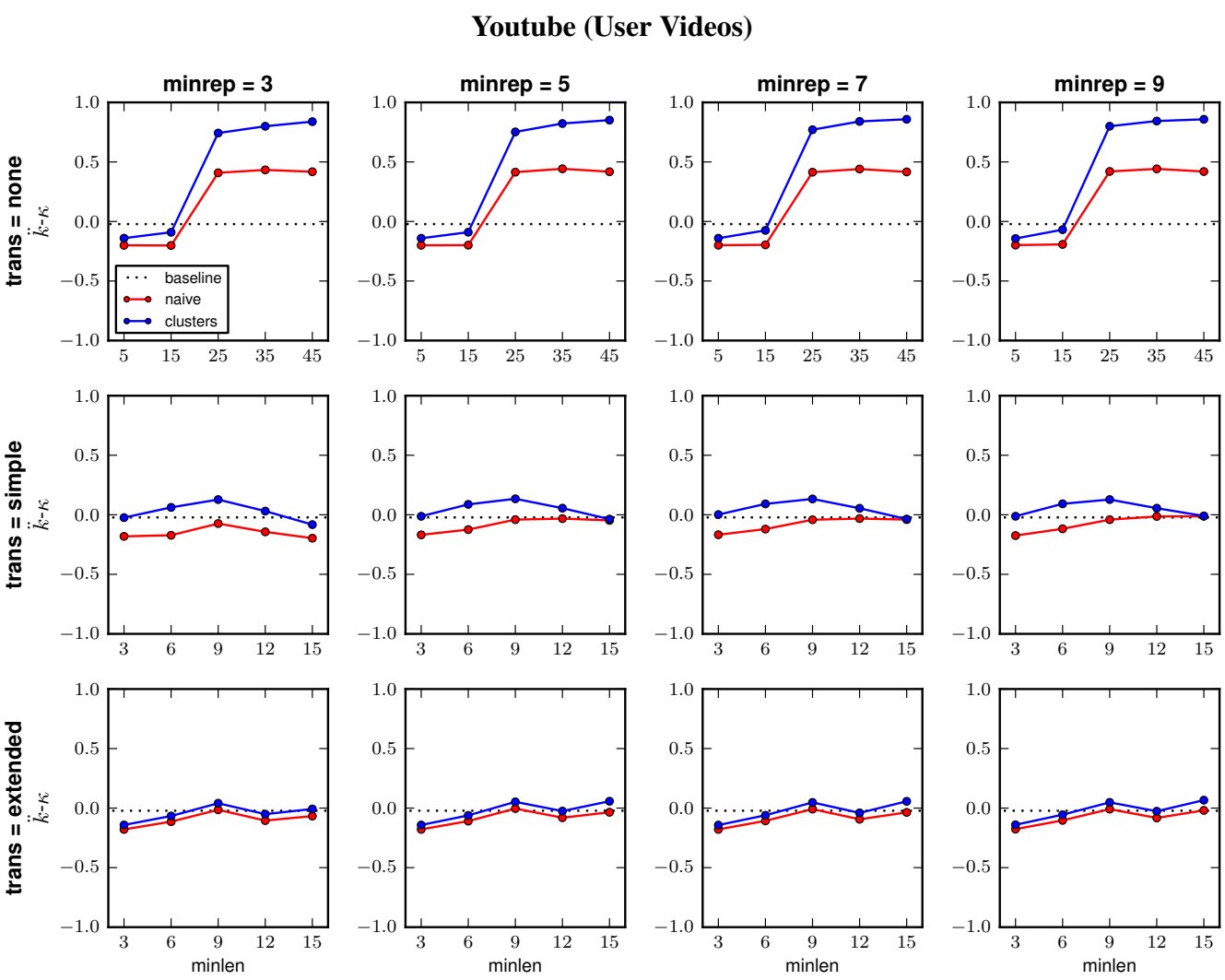

minrep = 3
minrep = 5
minrep = 7
minrep = 9
trans = none
trans = simple
trans = extended
$\ddot{k}$-κ
baseline
naive
clusters
minlen


Ebay (Grid)
minrep = 3
minrep = 5
minrep = 7
minrep = 9
trans = none
trans = simple
trans = extended
$\ddot{k}$-κ
baseline
naive
clusters
minlen


Google
minrep = 3
minrep = 5
minrep = 7
minrep = 9
trans = none
trans = simple
trans = extended
$\ddot{k}$-κ
baseline
naive
clusters
minlen


Reddit
minrep = 3
minrep = 5
minrep = 7
minrep = 9
trans = none
trans = simple
trans = extended
$\ddot{k}$-κ
baseline
naive
clusters
minlen


Youtube (Search Results)
minrep = 3
minrep = 5
minrep = 7
minrep = 9
trans = none
trans = simple
trans = extended
$\ddot{k}$-κ
baseline
naive
clusters
minlen


Youtube (User Videos)
minrep = 3
minrep = 5
minrep = 7
minrep = 9
trans = none
trans = simple
trans = extended
$\ddot{k}$-κ
baseline
naive
clusters
minlen