
# Character Recognition in Ancient Greek Papyrus

## Master Thesis

Gwenael Gendre

University of Fribourg

June 2021



# Abstract

The conservation of historical documents is very important, digitization helps to maintain them. Due to their possible degradation, handwriting recognition and thus complete transcription become difficult. One way to tackle this problem is Keyword Spotting, where all instances of a query keyword are retrieved. Graphs are a possible representation of handwriting and its variations and can therefore be used for Keyword Spotting.

This work focuses on a collection of ancient greek papyri, without word or character segmentation: we introduce a new graph representation, `Contour` graphs, and we successfully adapt a graph-based Keyword Spotting framework for segmentation-free use. We first test it on a well-studied document, the *George Washington* letters, and then on the papyri.

The experiences highlight two difficulties tied to segmentation-free graph matching on papyri: the complexity and needed time for keyword spotting greatly increase, not allowing us to fully utilize this framework. We also show that papyri are a really challenging document type, however a well-crafted binarization can almost double the obtained results.

Prof. Rolf Ingold, DIVA Research Group, Department of Informatics, University of Fribourg, Supervisor

Prof. Andreas Fischer, DIVA Research Group, Department of Informatics, University of Fribourg, Supervisor

Anna Scius-Bertrand, DIVA Research Group, Department of Informatics, University of Fribourg, Assistant

# Acknowledgements

I would like to warmly thank my supervisors, Prof. Rolf Ingold and Prof. Andreas Fischer, and Anna Scius-Bertrand from the DIVA group, who followed me, helped me and guided me during all of this work. I also thank Dr. Michael Stauffer, who agreed to share the code to the framework implemented in [1], Prof. Isabelle Marthot-Santaniello for the *ICDAR 2019* papyri along with their binarizations and their transcriptions, and Dr. Vlad Atanasiu for the manual reconstructions of the papyrus.

Many thanks go to my friends and family who encouraged me and stayed interested in my work, and above all to my girlfriend, for her help throughout this entire work, for her support during my highs and lows and for her unending patience for this work who took more of our time together than was necessary.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Keyword Spotting

The field of Pattern Recognition in Computer Science is concerned with the automatic recognition of patterns in data and then the use of those patterns to act on the data, *e.g.* to classify it [2]. Such computer algorithms are more powerful and efficient than humans for some tasks, for example Optical Character Recognition, the extraction of printed or handwritten text. Handwriting Recognition (HWR) is far more challenging than the recognition of machine printed text, mostly because the style and size of handwriting varies between the authors or even within the same author's different texts.

An interesting application of HWR is on historical documents: such documents have a priceless importance, both for their contents and for the fact that they are irreplaceable. To preserve them and protect them from degradations caused by the light, the air or the humidity, some of these documents are digitized. Once they are conserved as such, it is possible to make them available to a larger audience. However, an automatic full transcription of historical handwritten documents is not feasible, due to the variations in handwriting mentioned above and to the random noise caused by the nature of the document, such as stains or holes, ink fading, bleeding through or degrading...

To still be able to access, search and browse digitized handwritten documents, Keyword Spotting (KWS) has been used: the main idea is not to transcribe the whole document, but rather to find all instances of a query keyword and highlight them all for the user; KWS is more flexible and can tolerate errors. Many different methods for KWS exist, we will present the actual state of the art in the next Section of this work.

## 1.2 State of the art

We focus now on a brief overview of some KWS systems, using the classification followed by Stauffer *et al.* in [1]. They split a typical KWS process in three main steps:

(1) Preprocessing: the image documents are preprocessed to improve their quality by removing variations and noise on the image for better results. Two common steps when preprocessing are binarizing the documents, *i.e.* transforming the document into a binary image with black pixels as foreground and white as background, and filtering to remove noise.
Documents can then be segmented at the line-level or word-level, either manually with more time and effort or automatically. Other variations such as inclination of the text lines or the characters, size or scale of the words can also be corrected – or at least have their effects reduced.

(2) Formal Representation: once the preprocessing is done, the characteristics of the handwritten words, lines or documents are represented by a formal representation, a specific data structure such as feature vectors, strings, graphs, etc.

(3) Query: the formal representation of the query is compared to the unknown words in the document to retrieve all the instances of this query.

Three separations are made in the KWS classification in [1] are made as such: first, they identify two sorts of formal representations, either *statistical* or *structural*: statistical representations use feature vectors while structural representations use graphs (or strings or trees, who can be easily reduced to graphs). The next separation is for the querying algorithms: they can either be *template-based* or *learning-based*. Template-based algorithms compare the query's representation pairwise with a set of document images representations whereas the learning-based approach necessitates a priori training of a classifier model. The final separation is made between *query-by-example* (QbE) and *query-by-string* (QbS): in QbE, the query is a word image itself, while in QbS the keyword is queried via its text value.

We will now present some of the different parts of the classification with examples of their use.

Statistical representations can have different descriptors, where features can describe characteristics of the handwriting or general characteristics of the document (such as *scale-invariant feature transform* (SIFT) [3]), or be embeddings of the previous descriptors (*e.g. bag-of-visual-words* (BoVW) [4], *Fisher vectors* [5] or *pyramidal histogram of characters*(PHOC)[5]). Template-based approaches for statistical representations often use the sliding window approach with *e.g. dynamic time warping* (DTW) [6]; most of them only use QbE. The other approaches, learning-based approaches, can use *Hidden Markov Models* (HMMs) [6], neural networks and, more recently, *convolutional neural networks* (CNN) such as PHOCNet [7], R-PHOC [8] or CharNet [9]. Both template-based and learning-based approaches seem to prefer segmentation-based methods.

Structural representations are surprisingly rarer than their counterpart statistical representations, given how flexible graphs can be and how they seem to naturally suit handwriting representation. The possible reason for this is the complexity linked to graphs and their handling. Template-based approaches can use graph representations and a graph-matching algorithm, or the *inkball model* [10] which is another interesting possibility.

The handwritten historical documents have some additional limitations, such as their size and the cost of obtaining labeled training samples, which make CNNs harder to use on them; and considering the relatively low exploration of graph matchings, Stauffer *et al.* decided to study this particular direction. Given their good results, we chose to expand their work for segmentation-free: their graph-based approach segments datasets to match word images. However, not all datasets can be processed as such: the dataset that primarly interested us has no word separation in its text.

## 1.3 Contributions

In this thesis, we introduce two new contributions. The first novelty is a new graph representation, `Contour` graphs, that keeps the stroke width information. This representation is obtained by setting nodes with regular intervals on the characters' contour; Subsection 3.2.1 has the full explanation and the algorithm for creating `Contour` graphs.

The second contribution, the main scope of this work, is the adaptation of the framework created by Stauffer *et al.* in [1] for segmentation-free Keyword Spotting. The matching system was changed in order to extract small graphs from the document via windows sliding on a grid and then match these smaller graphs. This process is in explained in detail in Section 3.4.

This work will be organized as follows: first, the datasets used in this work will be presented in Chapter 2 with their preprocessing; the Chapter 3 addresses the whole graph subject: definitions, the graph representations used, graph matchings and edit distances, as well as the global functioning of this framework. Chapter 4 covers the experiments we made, the setups, the respective results and discussions about them. Finally, Chapter 5 concludes this work with a summary and possible future works on this topic.

# Chapter 2

# Datasets

This chapter will present the two datasets used in this work: the *George Washington* manuscript (GW)[1] and the *DIBCO 2019 Papyri Dataset*[2] and how they have been preprocessed.

## 2.1 Description

### 2.1.1 *George Washington* dataset (GW)

The *George Washington* (GW) dataset is a well-known and studied dataset. It consists of twenty pages of letters handwritten by George Washington and one of his associates in 1755. The writing is rather consistent, the document does not show major signs of degradation. The most notable variations are seen with respect to the words' scaling. The two first pages are displayed in Figure 2.1.

We will focus mainly on the complete pages, but the word images and line images are also available. The pages have a complete transcription, but the groundtruth only indicates the lines' position. This means that there is no precise word-level groundtruth for the pages.

### 2.1.2 *DIBCO Papyri* dataset

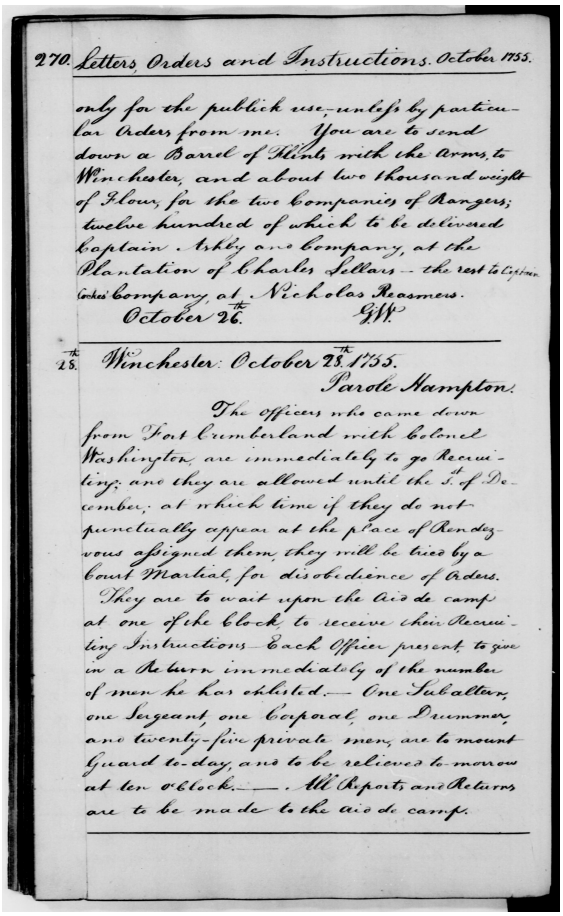
The *DIBCO Papyri dataset* was used for the *Competition on Document Image Binarization* (DIBCO 2019) as part of ICDAR 2019 [11]. It contains ten original images of papyri containing fragments of Homer's *Iliad* and their corresponding binarized version, built manually by members of the D-Scribes project[3]. Their original *Iliad* corpus contains around 1500 fragments, dating from the 3rd century BC to the 7th century AD. The papyri were chosen to reflect the diversity of literary papyri, therefore coming from various places and periods. In addition, they have different states of preservation and restoration. The images are provided by the institutions that own the papyri and are thus also heterogeneous. This dataset has visibly much more noise and degradation than GW, the characters' borders and strokes are less precise; these will be obstacles that the KWS will have to overcome. Two of the papyri can be found in Figure 2.2.

The groundtruth for this dataset had to be obtained manually: we received diplomatic transcriptions for the ten documents from Dr. Marthot-Santaniello, we then had to determine the character's bounding box and add it to the groundtruth file. This slow process meant that we limited ourselves to greek characters that could easily be recognized.

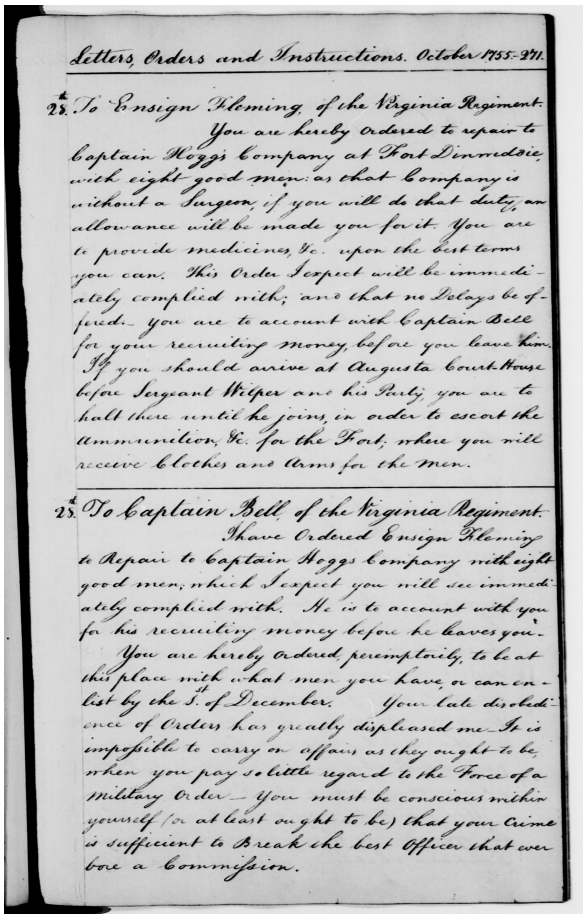
[1] George Washington Papers, Series 2, Letterbooks 1754 to 1799: Letterbook 1, pp. 270-279 & 300-309. Retrieved from the Library of Congress, www.loc.gov/item/mgw2.001/

[2] Papyri Dataset at ICDAR 2019 DIBCO, https://vc.ee.duth.gr/dibco2019/benchmark/

[3] D-Scribes, Ancient History, University of Basel, https://d-scribes.philhist.unibas.ch/en/home/

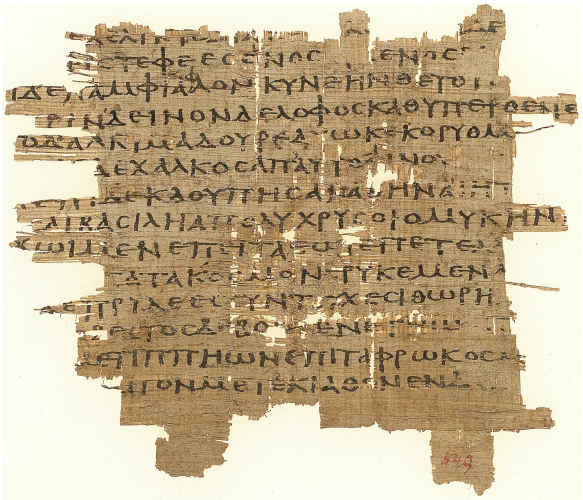



2.1.1: GW: page 270

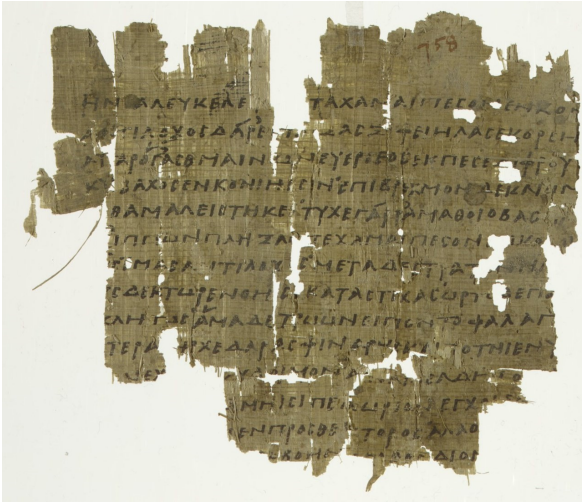



2.1.2: GW: page 271

Figure 2.1: Greyscale pages of the GW documents



2.2.1: Papyrus: *P.Corn. Inv. MSS. A 101. XIII*

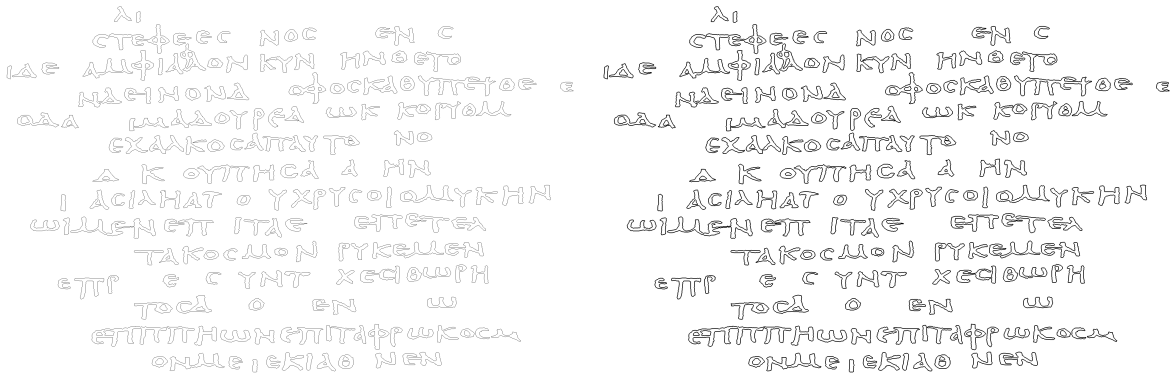


2.2.2: Papyrus: *P.CtYBR inv. 69*

Figure 2.2: DIBCO Papyri examples

### 2.1.3 Reconstructed Papyrus

After the first steps with the *Papyri* dataset, on which KWS confirmed itself not to be easy, we received reconstruction of the *P.Corn. Inv. MSS. A 101. XIII* papyrus that had been manually crafted by Vlad Atanasiu and show the contour of the characters with a width of one and two pixels, as shown in Figure 2.3.



2.3.1: Thin reconstruction

2.3.2: Bold reconstruction

Figure 2.3: Reconstructions for *P.Corn. Inv. MSS. A 101. XIII*

This document will serve as a comparison with the non-reconstructed papyrus, to see how much the noise can affect the results.

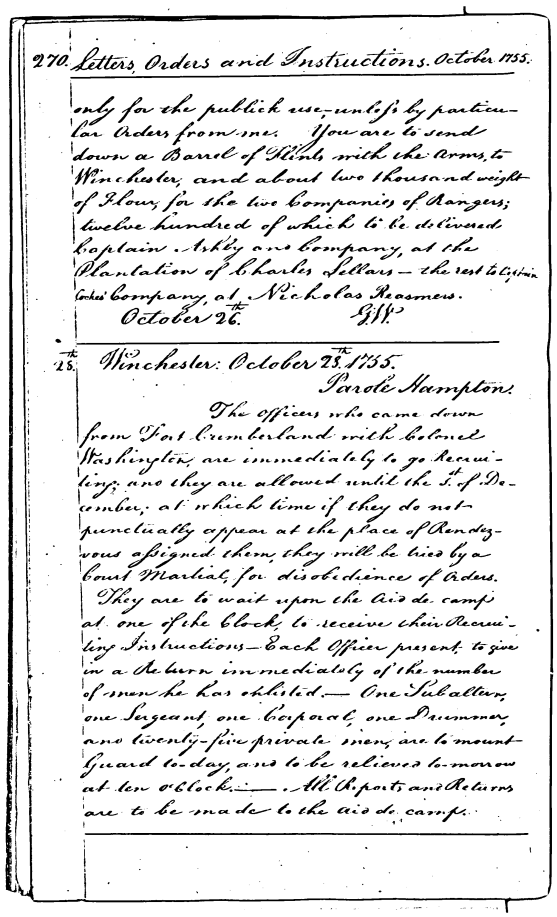
## 2.2 Image Preprocessing

The preprocessing steps of a document are important to ease the graph extraction from a document as much as possible. Unwanted variations can come from the degradation of the document (fading or smearing of the ink, stains or holes on the writing support) or from the image quality (bad scanning conditions) for example.

### 2.2.1 *George Washington*

The preprocessing for GW is fully explained in [1], we will quickly explain the important steps here. First, a *Difference of Gaussian* (DoG) filter is applied: the resulting image is obtained by substracting the convolution of the original image with a Gaussian of standard deviation $\sigma_2$ from convolution of the same image with another Gaussian of standard deviation $\sigma_1$ (with $\sigma_2 > \sigma_1$). A Gaussian filter blurs the image, so the Difference of Gaussian simply substracts a blurred image from a less-blurred image. This aims to get rid of the noise in the document and focuses on the document's important features. The document is then binarized: the intensity of every pixel in the grayscale image is compared to a global threshold; darker pixels (*i.e.* pixels with lower intensity) become black and part of the foreground, brighter pixels become white and part of the background. The first two pages binarized are shown in Figure 2.4.

For our implementation of Segmentation-free KWS, the preprocessing on GW stopped here. However, we also tested our graph representations on word images, these had been obtained by segmenting the documents into line images and then word images and by finally correcting the skew, *i.e.* the inclination of the document.



270 Letters, Orders and Instructions. October 1755.

only for the publick use, unless by particular Orders from me. You are to send down a Barrel of Flints with the Arms, to Winchester, and about two thousand wight of Shot, for the five Companies of Rangers; twelve hundred of which to be delivered Captain Ashby and Company, at the Plantation of Charles Sellars — the rest to Captain Cocks Company, at Nicholas Reasmes.

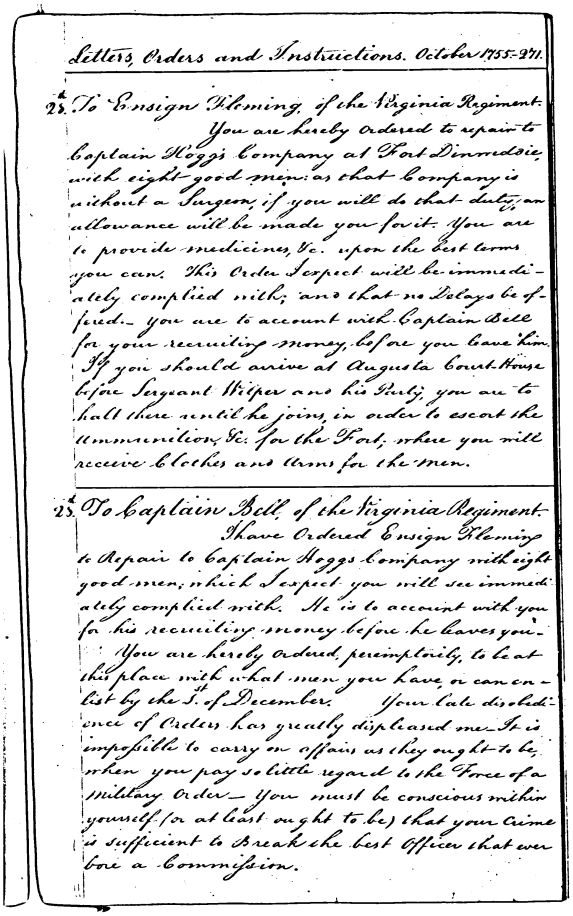
October 26th 1755

25th Winchester: October 25th 1755.

Parole Hampton.

The Officers who came down from Fort Cumberland with Colonel Washington, are immediately to go Recruiting; and they are allowed until the 5th of December, at which time if they do not punctually appear at the place of Rendezvous assigned them, they will be tried by a Court Martial, for disobedience of Orders.

They are to wait upon the Aid de camp at one of the Clock to receive their Recruiting Instructions — Each Officer present to give in a Return immediately of the number of men he has enlisted. — One Subaltern, one Sergeant, one Corporal, one Drummer, and twenty-five private men, are to mount Guard to-day, and to be relieved to-morrow at ten o'Clock. — All Reports and Returns are to be made to the Aid de camp.



Letters, Orders and Instructions. October 1755-271.

25th To Ensign Fleming, of the Virginia Regiment.

You are hereby ordered to repair to Captain Hoggs Company at Fort Dinwiddie, with eight good men: as that Company is without a Surgeon, if you will do that duty, an allowance will be made you for it. You are to provide medicines, &c. upon the best terms you can. This Order I expect will be immediately complied with; and that no Delays be offered. — You are to account with Captain Bell for your recruiting money, before you leave him. If you should arrive at Augusta Court House before Sergeant Wiper and his Party, you are to halt there until he joins, in order to escort the Ammunition, &c. for the Fort, where you will receive Cloaths and Arms for the men.

25th To Captain Bell, of the Virginia Regiment.

I have Ordered Ensign Fleming to Repair to Captain Hoggs Company with eight good men, which I expect you will see immediately complied with. He is to account with you for his recruiting money before he leaves you.

You are hereby Ordered, peremptorily, to be at this place with what men you have, or can enlist by the 5th of December. Your late disobedience of Orders has greatly displeased me. It is impossible to carry on affairs as they ought to be, when you pay so little regard to the Force of a Military Order — You must be conscious within yourself (or at least ought to be) that your Crime is sufficient to Break the best Officer that ever bore a Commission.

2.4.1: GW: page 270

2.4.2: GW: page 271

Figure 2.4: Binarized pages of the GW documents

### 2.2.2 Papyri binarization

The *Papyri* dataset contains both the original and the binarized documents, which served as groundtruth for the DIBCO 2019. We decided to try to obtain our own binarizations and compare them with the groundtruth binarizations. We will now present the different steps we tried.

Our first step was to convert the images from color to grayscale, and, to speed up the calculations, to work only on small patches, here of size 500 by 500 pixels. Figure 2.5 shows such a patch.

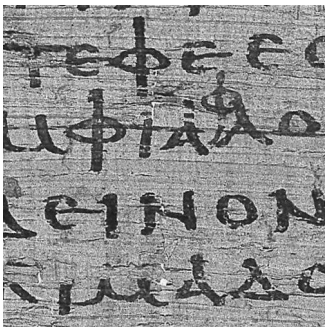


Figure 2.5: Grayscale patch

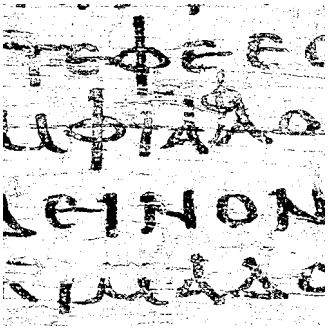
Directly binarizing the documents with a global threshold was not sufficient: due to the structure of papyrus, some parts will be darker than the ink and will thus stay in the binarized image, see Figure 2.6. In these figures, binarizations with three different thresholds are shown. We observe that trying to remove the marks by lowering the threshold will take an important part of the characters' ink away but not remove said marks.

The next step has then been to use filters to get rid of the noise caused by the papyrus itself and focus on the image's important features. The first filter is the Difference of Gaussians (DoG): the resulting image is obtained by substracting the convolution of the original image with a Gaussian of standard deviation $\sigma_2$ from convolution of the same image with another Gaussian of standard deviation $\sigma_1$ (with $\sigma_2 > \sigma_1$). A Gaussian filter blurs the image, so the Difference of Gaussian simply substracts a blurred image from a less-blurred image.

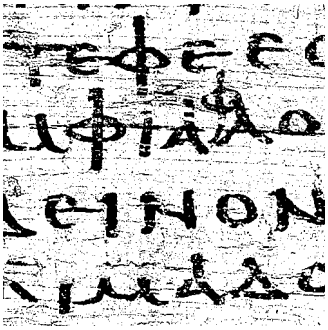
We used Python with OpenCV[4] to edit all the pictures. At first,we implemented the Difference of Gaussians with the usual NumPy operations, but it did not produce the expected results because the Numpy and OpenCV libraries do not treat the addition of images the same way: OpenCV has a saturated operation where NumPy uses a modulo operation, letting values "jump" between 0 and 255. This is not what we needed: subtracting pixels with almost equal values could lead to very high as well as very low values next to each other. We see in Figure A.1 the difference between the two methods: the saturated substraction effectively keeps the important features of the image[5], where the modulo substraction creates patterns of black and white spots on apparently

[4]https://opencv.org/

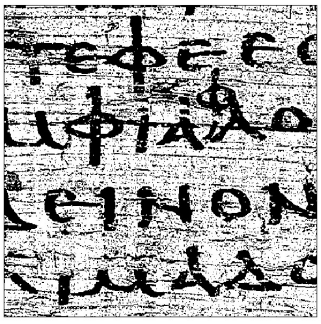
[5]Source of the picture: https://upload.wikimedia.org/wikipedia/commons/d/da/Flowers_before_difference_of_gaussians.jpg



2.6.1: Binarization with threshold 70

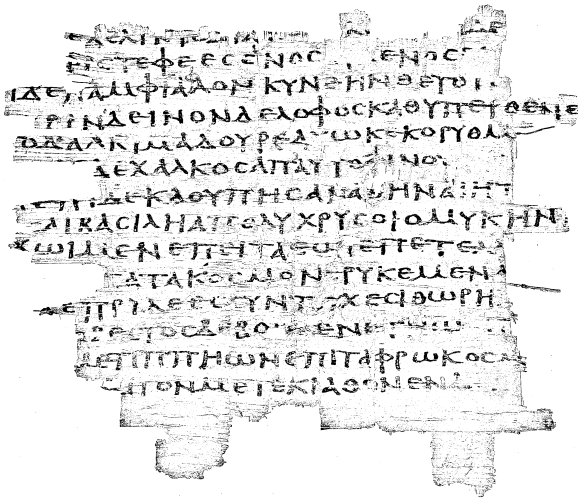


2.6.2: Binarization with threshold 100

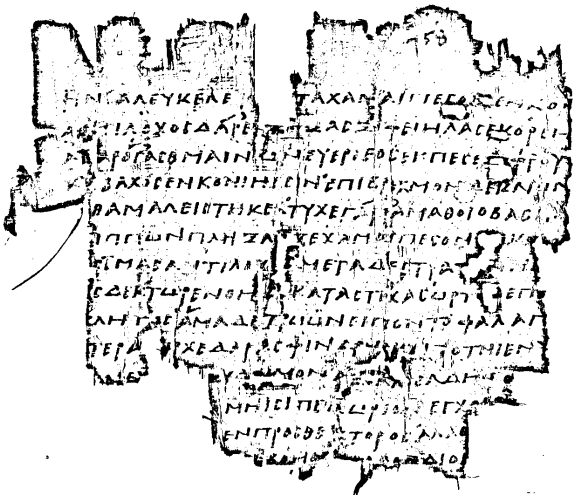


2.6.3: Binarization with threshold 130

Figure 2.6: Comparison of different thresholds for binarization



2.7.1: Threshold 210, $\sigma_1 = 0.5, \sigma_2 = 40$



2.7.2: Threshold 240, $\sigma_1 = 0.5, \sigma_2 = 10$

Figure 2.7: Difference of Gaussians binarization

plain surfaces. Therefore, OpenCV's method is the one we used to implement a correct Difference of Gaussians.

Our DoG implementation uses two parameters, the standard deviation for each Gaussian kernel. All combinations of $(\sigma_1, \sigma_2) \in \{0.1, 0.5, 1\} \times \{10, 40\}$ are tested on the ten patches. Higher $\sigma$ values gave out too blurry images to be exploitable. Figure A.2 shows an example comparison between all pairs of parameter on a single patch.

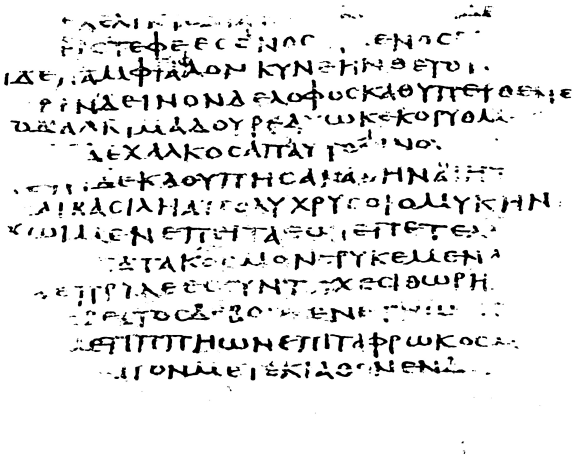
The standard deviation $\sigma_1$ does not appear to influence the resulting DoG as much as $\sigma_2$ does. We decided to simplify and keep 0.5 as the only $\sigma_1$ value. It is less clear whether there is a "best" value for $\sigma_2$. The characters seem to be more discernible with $\sigma_2 = 40$, but black markings appeared around the edges and the eventual holes of the papyrus, as seen on Figures A.2.4 to A.2.6. Interestingly, these marks did not appear on two of the ten documents.

The initial patch selection was aimed at having as much text as possible on the patches, but this caused some patches to not have holes or edges of the papyrus on them. In order not to miss any interesting result or artifact due to mispositioned patches, we decided to work on the complete documents again, at the cost of a small time increase.

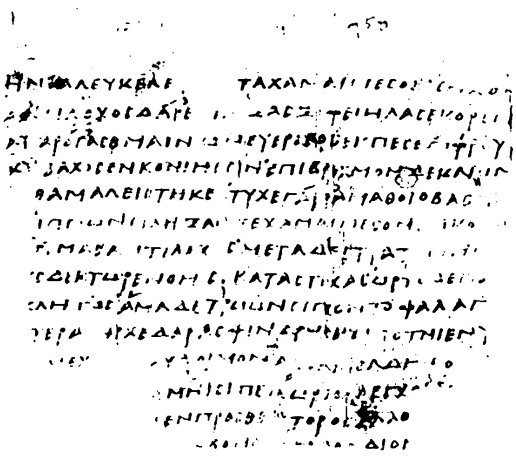
At this step, we realized that the Difference of Gaussians did not eliminate the small documents' impurities as well as we hoped, but instead detected them as important features. This was confirmed by the binarization tests: for each document, two Difference of Gaussians images ($\sigma_1 = 0.5, \sigma_2 \in \{10, 40\}$) were binarized with different thresholds. This time, good results were only obtained with very high thresholds. In comparison to the direct binarization where the best thresholds ranged between 50 and 150, here no good image was produced with thresholds under 200. We then chose visually the best DoG binarization for each document and made the following observations: the marks caused by the papyrus itself stayed on all images, but with different intensities; as seen in the direct binarizations, a lower threshold made the characters disappear as fast as these marks; and finally that the Difference of Gaussians was apparently not a suitable filter for the style of documents we were working on.

Below are shown two interesting examples: in Figure 2.7.1, we can observe the recurring problem of the markings and the characters having the same intensity: to eliminate the first, the latter also has to disappear. In Figure 2.7.2, another quality of papyrus means that such marks are less numerous, but the edges of the document have a thick black border.

We chose the median blur filter as the next filter to test because we supposed that blurring would help taking away the small markings and that the characters were big enough to stay well defined. The median blur filter uses a square kernel and assigns to each pixel the median value of the square, the parameter controls here the size of the kernel. We used three small kernel sizes, 5, 7 and 9, on all documents, bigger kernels blurred the images too much.



2.8.1: Median Blur of size 9, threshold 100



2.8.2: Median Blur of size 5, threshold 90

Figure 2.8: Median Blur Binarizations

There was no global best size on all documents, the size and the thickness of the characters and the document seemed to influence the results: a too small kernel on big characters would not eliminate well enough all the undesirable marks and a kernel too big relatively to the characters would blur them too much. Even then, only a few documents had a clear optimal kernel size. Therefore, all documents were blurred with all three kernel sizes and then binarized with different thresholds. This time, the thresholds were lower again: all best binarizations occurred between 70 and 100. We directly observed good results for almost all documents! Some images may even have been good enough to be used directly and were almost as good as the binarized groundtruth received with the documents. The Figure 2.8.1 shows such an example, it was obtained with a simple median blur kernel and a threshold binarization, it is clearly better than its DoG counterpart in Figure 2.7.1.

We can also compare Figure 2.7.2 with Figure 2.8.2: the characters are a bit rounded, but the overall quality stays very good.

However, not all documents produced such good results. For example, Figure A.3 shows us that there is not always a perfectly good threshold: in Figure A.3.1, the lower left part is not visible enough, but a higher threshold would be too high on the top right part, see Figure A.3.2.

A more extreme version is seen in Figure A.4. This is caused by the intensity difference in the original document: a simple binary threshold cannot be sufficient in this case.

One possible solution for this intensity difference would be to use an adaptive threshold. OpenCV has two available methods, Mean Adaptive Thresholding and Gaussian Adaptive Thresholding. Both methods compute the threshold value for each pixel (instead of using the same value for the whole image). The Mean method simply uses the mean of the neighbourhood region values as the threshold and the Gaussian one uses a weighted sum of the neighbourhood values, with Gaussian weights.

We used both methods on two documents that had problems with binary thresholds (see Figures A.3 and A.4). The results were not really better: the same black spots seen with the Difference of Gaussians appeared here again. Different parameters combinations were tried, the best results with adaptive thresholding are found in Figures A.5.1 and A.6.1. These images can be compared with the best DoG results in the Figures A.5.2 and A.6.2.

Finally, the last step was to compose the median blur and the Difference of Gaussians, to see if the median blur could be further improved. The resulting images looked very much alike, whether the median blur filter was used first or second, we saw no noticeable difference on any document.

The images obtained with this composition of filters were of globally better quality than those produced only with the Difference of Gaussians, but not better than the median blur images. Some of them were of equal quality, some of them were clearly worse.

After our experiments, using the median blur filter on images and then binarizing them seems to be the best solution restricted to the use of direct filters. More complex methods produce better results, the rest of this work will use the groundtruth binarizations manually obtained for the DIBCO with no further preprocessing. Figure 2.9 shows one binarization, to be compared to the reconstruction in Figures 2.3.1 and 2.3.2

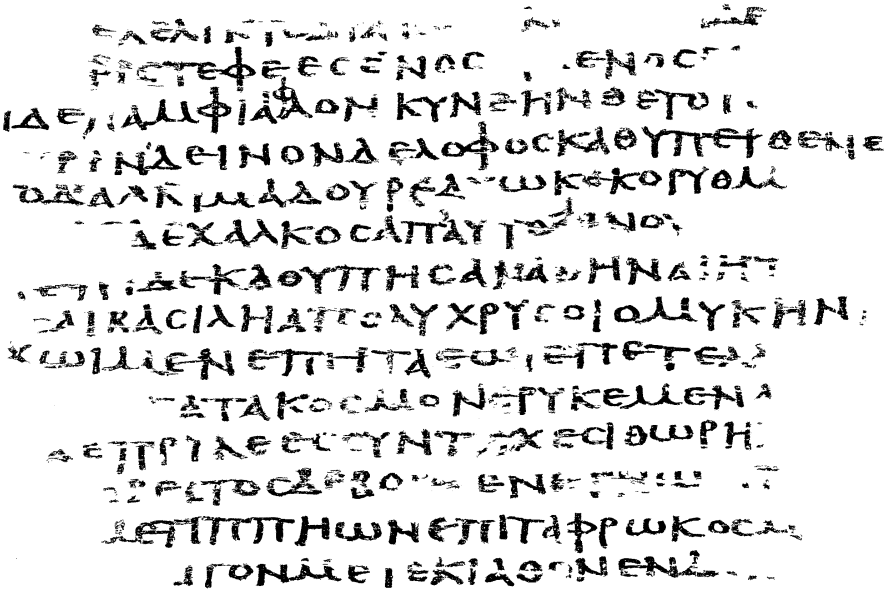


Figure 2.9: Binarization for *P.Corn. Inv. MSS. A 101. XIII*

# Chapter 3

# Methods

As said in Chapter 1, graphs are an interesting way to represent handwritten documents for Keyword Spotting. This chapter will address the basic definitions of graphs in Section 3.1, Section 3.2 will focus on the two graph representations used in this work and Section 3.3 will be on the subject of graph matchings.

## 3.1 Graph definitions

**Definition 3.1.** *Graph* A graph $g$ is a four-tuple $g = (V, E, \mu, \nu)$ where

- $V$ is the finite set of vertices
- $E$ is the finite set of edges
- $\mu : V \to L_V$ is the vertex labelling function
- $\nu : E \to L_E$ is the edge labelling function.

Two further separations can be made for graphs, based on the direction of their edges or on their labelling functions. Directed edges between vertices create *directed* graphs, undirected edges create *undirected* graphs. The other categorization is between *unlabelled* and *labelled* graphs. Unlabelled graphs have no labels, that is, the label alphabets are empty ($L_V = L_E = \emptyset$). Labels on vertices and edges can take many forms: litterals, symbols or vectors.

**Definition 3.2.** *Subgraph*: A graph $g_1 = (V_1, E_1, \mu_1, \nu_1)$ is a subgraph of $g_2 = (V_2, E_2, \mu_2, \nu_2)$, written $g_1 \subseteq g_2$, if

(1) $V_1 \subseteq V_2$

(2) $E_1 \subseteq E_2$

(3) $\mu_1(v) = \mu_2(v)\ \forall v \in V_1$

(4) $\nu_1(e) = \nu_2(e)\ \forall e \in E_1$.

An *induced subgraph* is a subgraph created by removing some vertices and only removing edges adjacent to said vertices. The condition (2) above is thus replaced by

(2′) $E_1 = E_2 \cap V_1 \times V_1$.

## 3.2 Graph representations

The preprocessed and binarized documents have to be transformed into graphs in order to be analyzed. These graphs should be able to hold the important characteristics of an image while still reducing the amount of information to treat. This work will focus on Contour Graphs, a representation that, to my knowledge, has not yet been used in Graph-Based Keyword Spotting.

### 3.2.1 Contour graphs

The idea behind this representation, that will be denoted `Contour` from now on, is to keep the stroke width when creating a graph. The four representations proposed in [1] do not use this feature, which may convey important metadata about the character and help recognize the writer.

The graphs are extracted from binarized documents as follows: first, obtain the coordinates of the contours. This was done with the corresponding function of the *OpenCV* library [1]. Equidistant points of distance $D$ are selected from each contour component as the vertices, edges are added between consecutive vertices. This simple procedure is explained in Algorithm 1.

**Algorithm 1** Contour graphs extraction

**Input:** Binarized image $B$, Distance threshold $D$, Approximation value $\varepsilon$
**Output:** Graph $g = (V, E)$ with vertices $V$ and edges $E$

**function** CONTOUR($B, D, \varepsilon$)
  **for** Each contour component $C \in B$ **do**
    $V_C = \{(x, y) \in C \mid (x, y)$ are points in equidistant intervals $D\}$
    $E_C = \{(u, v) \in V \cap C \mid (u, v)$ are consecutive vertices in $V \cap C\}$
    **if** $\varepsilon \neq 0$ **then**
      $(V_C, E_C) \leftarrow$DOUGLAS-PEUCKER$((V_C, E_C), \varepsilon)$
  **return**

Varying $D$ allows to change the size of the graph, but too distant nodes may cause the final graph to be too far from the original character. To keep a reasonable size without loosing too much information, we used the Douglas-Peucker algorithm to simplify the contours. Algorithm 2 shows the pseudo-code. The Douglas-Peucker algorithm recursively simplifies a polyline as follows: select the first and last nodes as boundaries and find the node the furthest away from the line segment between the boundaries. If the node is further than the approximation value $\varepsilon$, then call the algorithm recursively on the two parts of the polyline: from the first boundary to the distant node and from the distant node to the second boundary. If all nodes are close enough, erase them all and only keep the two boundary points.The *OpenCV*'s implementation of this algorithm was used.

**Algorithm 2** Douglas-Peucker Algorithm

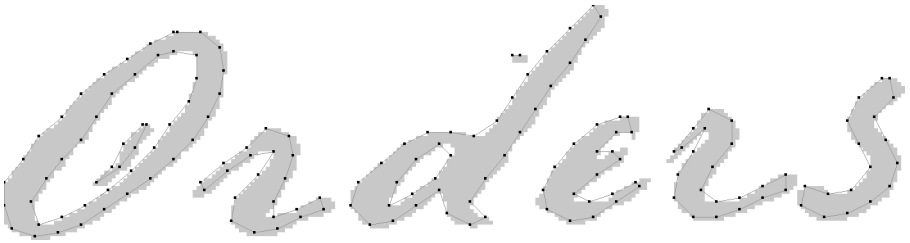
**Input:** Polyline $L = (P_1, ..., P_n)$ , Approximation value $\varepsilon$
**Output:** Polyline $L' \subseteq L$

**function** DOUGLAS-PEUCKER($L, \varepsilon$)
  Find point $P_i$ furthest from line segment $\overline{P_1 P_n}$
  **if** Distance between $P_i$ and $\overline{P_1 P_n} > \varepsilon$ **then**
    $L_1 \leftarrow$ DOUGLAS-PEUCKER$((P_1, ..., P_i), \varepsilon)$
    $L_2 \leftarrow$ DOUGLAS-PEUCKER$((P_i, ..., P_n), \varepsilon)$
    $L' \leftarrow L_1 + L_2$
  **else**
    $L' \leftarrow \overline{P_1 P_n}$
  **return** $L'$

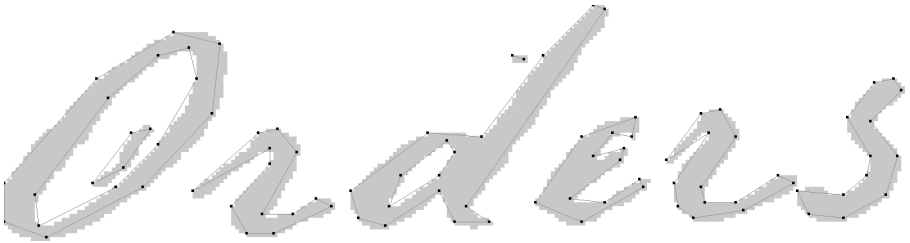
Figure 3.1 shows one word of GW, Figure 3.2 shows two examples of `Contour` graphs with and without the Douglas-Peucker algorithm, the Figures A.8 and A.9 have more parameters. Two papyri characters of different sizes (see Figures A.10 and A.13) and their `Contour` graphs are compared in Figures A.11, A.12, A.14 and A.15.

[1] Open Source Computer Vision Library, `https://opencv.org/`

Figure 3.1: Binarized word of GW



3.2.1: `Contour` graph, $D = 6$



3.2.2: `Contour` graph, $D = 1, \varepsilon = 2$

Figure 3.2: `Contour` graphs on *Orders*, without and with Douglas-Peucker

### 3.2.2 Keypoint graphs

The keypoint graph representation was introduced in [12], we follow the extended and refined version explained in [1], where it was the best performing representation on the George Washington dataset using the Hausdorff Edit Distance algorithm. From now on, this representation will be referred to as `Keypoint`.

The pseudocode is found in Algorithm 3 and comes directly from [1], however, we used our own implementation of the code to generate the `Keypoint` graphs.

First, the binarized document is skeletonised, in our code with the function from the *sciki-image* library[2], the skeleton is then separated into connected components. The keypoints, *i.e.* the end points and the junction points, are extracted from each connected component. End points are detected as pixels with exactly one or no neighbouring pixels, junction points are pixels with more than two neighbouring pixels. These criteria are not sufficient for circular structures, *e.g.* the letter "o", an arbitrary pixel is chosen as a junction point in such cases.

The thinning algorithm, proposed in [13], may have neighbouring keypoints: we make sure to only keep endpoints with no other endpoints in their 8-neighbourhood, and only keep junction points that have no neighbouring junction points with higher connectivity next to them. To be noted that this is not mentioned in the pseudocode.

Next, the selected junction points are removed from the connected components, transforming them into connected subcomponents. For each connected subcomponent, equidistant intermediate points of distance $D$ are added by starting at an end of the connected subcomponent and counting the steps to all other points. In the ideal case, connected subcomponents would only form segments, however, due to the skeleton generation and the fact that not all junction points have been removed in the previous step, circular structures can still

[2]scikit-image, `https://scikit-image.org/`

appear. This problem is solved as before by taking an arbitrary pixel as the starting point.

The final step is to add the edges to the graph. If vertices $u, v \in V$ are directly connected by a chain of pixels in the skeleton $S$, an undirected edge $(u, v)$ is added to $E$.

**Algorithm 3** Keypoint-based graph extraction

**Input:** Skeleton image $S$, Distance threshold $D$
**Output:** Graph $g = (V, E)$ with vertices $V$ and edges $E$

**function** KEYPOINT($S, D$)
  **for** Each connected component $CC \in S$ **do**
    $V = V \cup \{(x, y) \in CC \mid (x, y)$ are end points or junction points$\}$
    Remove junction points from $CC$
    **for** Each connected subcomponent $CC_{sub} \in CC$ **do**
      $V = V \cup \{(x, y) \in CC_{sub} \mid (x, y)$ are points in equidistant intervals $D\}$
  **for** Each pair of vertices $(u, v) \in V \times V$ **do**
    $E = E \cup (u, v)$ if the corresponding points are connected in $S$
  **return** $g$

Figures 3.3 show a `Keypoint` graph obtained on *Orders*, Figures A.16, A.17 and A.18 show the `Keypoint` graphs obtained on the same binarizations as in 3.2.1, again with three different distance parameters $D$.

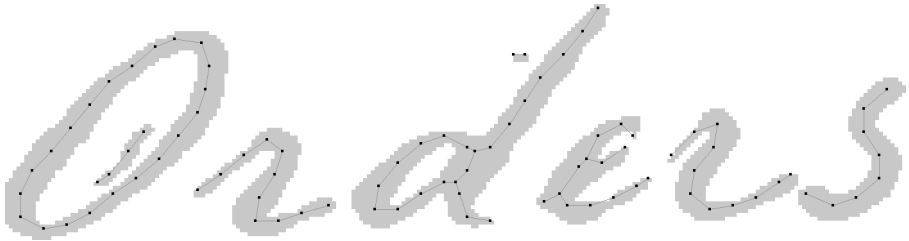


Figure 3.3: `Keypoint` graph on *Orders*

## 3.3 Graph matchings

This section follows closely the argumentation in [1].

Once we have represented our documents as graphs, in order to perform KWS on them, we must be able to compute how similar the graphs are, here our query graph and the different queried target graphs. This is made with a *graph matching* algorithm: it tries to match similar substructures of two graphs together and allowing us to get the similarity (or dissimilarity) measure of the two graphs. There are two types of graph matchings: *exact graph matchings* and *inexact graph matchings*.

Exact graph matchings have strong conditions: they will only map two graphs together if the mapping respects the edge structure and the labelling of the graphs. Inexact graph matchings are more flexible and allow for some error-tolerance: mappings may violate the edge structure, nodes or edges with different labels may be mapped together, and nodes and edges may be inserted and/or deleted.

Variations will always happen when creating graphs for KWS, it can be from the degradation of the document itself or from the differences in the handwriting, *e.g.* the word scale or some interwriter variation. Inexact graph matchings can handle such cases where graphs are close but no perfect mapping can be found, they will therefore be the matchings used for the rest of this work.

To measure dissimilarity, inexact graph matchings can assign a cost to the different nodes and edges operations: mapping (or substitution), insertion and deletion; minimising the sum of costs of all operations leads to finding the best matching. Such a minimisation has been found to be NP-complete, we must thus use sub-optimal inexact graph matching paradigms if we want to keep polynomial - and not exponential - time. The solution is to use the *graph edit distance* (GED) [14, 15].

### 3.3.1 Graph Edit Distance (GED)

The *graph edit distance* (GED) is a very flexible graph matching model. GED has been adapted from the string edit distance, here a graph $g_1 = (V_1, E_1, \mu_1, \nu_1)$ can be *edited* into $g_2 = (V_2, E_2, \mu_2, \nu_2)$ with following operations: substitution, deletion and insertion of both nodes and edges, formally described as:

(1) Node substitution $u \in V_1$ and $v \in V_2$ denoted by $(u \to v)$ ,

(2) Node deletion $u \in V_1$ denoted by $(u \to \varepsilon)$ ,

(3) Node insertion $v \in V_2$ denoted by $(\varepsilon \to v)$ ,

where $\varepsilon$ is the empty node; edge edit operations are defined similarly. Every edit operation $e$ is assigned a meaningful cost $c(e)$. An *edit path* is a set of operations that transforms $g_1$ into $g_2$. The *graph edit distance* (GED) $d_{\lambda_{min}}(g_1, g_2)$ between $g_1$ and $g_2$ is defined by

$$d_{\lambda_{min}}(g_1, g_2) = \min_{\lambda \in \Upsilon(g_1, g_2)} \sum_{e_i \in \lambda} c(e_i) \ ,$$

where $\Upsilon(g_1, g_2)$ denotes the set of all edit paths transforming $g_1$ into $g_2$, $c$ is the cost function measuring the weight $c(e_i)$ of the edit operation $e_i$ and $\lambda_{min}$ refers to the minimum cost path found in $\Upsilon(g_1, g_2)$.

However, GED is computationally complex and finding the exact minimum $\lambda_{min}$ requires an exponential time. For this reason, we use suboptimally approximate GED with the *Hausdorff edit distance* (HED)[16].

### 3.3.2 Hausdorff Edit Distance (HED)

The *Hausdorff edit distance* (HED) is a lower bound approximation of GED with quadratic time complexity. The GED problem is transformed into a set matching problem, in an analog way to the Hausdorff distance for finite sets: every node of the source graph is compared to every node of the target graph. In order not to be too sensitive to variations in handwriting and possible outliers, the maximum is replaced by a sum and the modified Hausdorff distance $H'$ between two sets $A = \{a_1, ..., a_n\}$ and $B = \{b_1, ...b_n\}$ is defined by [17]

$$H'(A, B) = \sum_{a \in A} \min_{b \in B} d(a, b) + \sum_{b \in B} \min_{a \in A} d(a, b) \ .$$

For $g_1 = (V_1, E_1, \mu_1, \nu_1), g_2 = (V_2, E_2, \mu_2, \nu_2)$ two graphs, the HED is given by [16] as

$$HED(g_1, g_2) = \sum_{u \in V_1} \min_{v \in V_2 \cup \{\varepsilon\}} f(u, v) + \sum_{v \in V_2} \min_{u \in V_1 \cup \{\varepsilon\}} f(u, v) \ ,$$

where $f(u, v)$ is a cost function for nodes matchings defined as

$$f(u, v) = \begin{cases} c(u \to \varepsilon) + \sum_{i=1}^{|P|} \frac{c(p \to \varepsilon)}{2} & \text{for nodes deletions } (u \to \varepsilon) \\ c(\varepsilon \to v) + \sum_{i=1}^{|Q|} \frac{c(\varepsilon \to q)}{2} & \text{for nodes insertions } (\varepsilon \to v) \\ \frac{c(u \to v) + \frac{HED(P,Q)}{2}}{2} & \text{for node substitutions } (u \to v) \ , \end{cases}$$

where $P = \{p_1, ..., p_{|P|}\}$ and $Q = \{q_1, ..., q_{|Q|}\}$ are the sets of edges adjacent to $u$ and $v$.

The edge edit costs are implicitly given by the node substitutions:

$$HED(P, Q) = \sum_{p \in P} \min_{q \in Q \cup \{\varepsilon\}} g(p, q) + \sum_{q \in Q} \min_{p \in P \cup \{\varepsilon\}} g(p, q) \ ,$$

where $g(p, q)$ is the cost function for edge matchings:

$$g(p, q) = \begin{cases} c(p \to \varepsilon) & \text{for edge deletions } (p \to \varepsilon) \\ c(\varepsilon \to q) & \text{for edge insertions } (\varepsilon \to q) \\ \frac{c(p \to q)}{2} & \text{for edge substitutions } (p \to q) \ . \end{cases}$$

## 3.4 Segmentation-free Keyword Spotting

All the individual steps described until now can be merged into a segmentation-free graph-based keyword spotting framework.

As explained in Section 2.2, we used already preprocessed binarized documents. Graphs are then extracted from the images using two different algorithms yielding two different representations, `Contour` and `Keypoint` graphs (see Section 3.2). The nodes and edges of the `Contour` graphs represent the contour of a character, while `Keypoint` graphs keep the most significant nodes of a character's skeleton. At first, the nodes correspond to the pixel coordinates; to allow a more meaningful comparison, their $(x, y)$ coordinates are then normalized with following formula:

$$\hat{x} = \frac{x - \mu_x}{\sigma_x}, \ \hat{y} = \frac{y - \mu_y}{\sigma_y}$$

where $x, y$ are the original node position, $\hat{x}, \hat{y}$ are the updated node position, and $(\mu_x, \sigma_x)$ and $(\mu_y, \sigma_y)$ respectively denote the mean and standard deviation of all $x$- and $y$-coordinates of the considered graph.

The next step is to match the query graphs with the target document graphs. Since there is no segmentation, the target graphs have to be extracted from the bigger document graphs first. For each query graph, windows of different sizes are slid over each target document on a grid pattern with regular horizontal and vertical steps, the document graph is cropped to produce the target graphs; the different windows' dimensions are proportional to the query image and should vary accordingly with the relative scales of all documents. A simple selection of the target graphs is made: a maximum ratio $n$ is chosen, the target graph is rejected if the actual ratio $r$ between the number of nodes in the query graph and in the target graph is too far from 1, *i.e.* if

$$r \leq \frac{1}{n} \text{ or } r \geq n \ .$$

We will now present the cost model used for our GED and its faster approximation, HED, a flexible model adapted to handwriting. This model uses four parameters, the first one is $\alpha \in [0, 1]$, it represents the relative importance of the node edit operations and the edge edit operations. More precisely, the cost of a node operation is multiplied by the weight $\alpha$ and the cost of an edge operation is multiplied by the weight $(1 - \alpha)$; a value of 0.5 means an equal balance between costs. The insertion and deletion operations have unique constant costs, $\tau_\nu \in \mathbb{R}^+$ is the cost of node operations and $\tau_e \in \mathbb{R}^+$ is for edge operations. The cost of substitutions should depend on the dissimilarity of the labels, here the nodes' labels are the $(x, y)$-coordinates, we will thus use a weighted Euclidean distance for this. The cost of a node substitution $(u \to v)$ with $\mu_1(u) = (x_u, y_u)$ and $\mu_2(v) = (x_v, y_v)$ is computed as follows:

$$c(u \to v) = \sqrt{\beta(\sigma_x(x_u - x_v))^2 + (1 - \beta)(\sigma_y(y_u - y_v))^2}$$

where $\sigma_x$ and $\sigma_y$ denote the standard deviation of the node coordinates in the query graph, and $\beta \in [0, 1]$ is the weighting parameter determining the balance between $x-$ and $y-$coordinates. Table 3.1 summarizes the four cost function parameters.

| Parameter | |
|---|---|
| $\alpha$ | Weighting parameter between node and edge operation costs |
| $\beta$ | Weighting parameter between $x-$ and $y-$coordinates |
| $\tau_v$ | Constant cost for node insertion and deletion |
| $\tau_e$ | Constant cost for edge insertion and deletion |

Table 3.1: Cost function parameters

Once the distances between the source graphs and all target graphs are known, they are then normalized over each target page. The differences in the groundtruths mean that the spotting results have to be computed differently. For GW, we only know the content of a line and the coordinates in the form of a bounding polygon, we therefore cannot precisely pinpoint words within the line. The bounding polygon is replaced by its bounding rectangle for an easier computation of intersection of the line and the source word; the word is considered in said line if the ratio of the area of the intersection over the area of the original source word is over a fixed threshold. Here we used a value of 0.9, and we checked whether the word appears in the line transcription to determine

if the match is correct or not. Proceeding like this does not give precise results, a meaningful comparison to previous works will be harder to obtain. On the Papyri, since we manually determined rectangular bounding boxes for some of the characters, we can directly compare them to the target window. Here, we use the ratio of the *Intersection over Union* areas, a ratio over 0.75 signified an overlap of the characters.

To evaluate the results of our models, we need to quantify how good they are, *i.e.* how correct predictions are. We first count the *true positives* ($TP$), *false positives* ($FP$), *false negatives* ($FN$) and *true negatives* ($TN$). The confusion matrix resumes these four values in Table 3.2:

| | | **Prediction** | |
|---|---|---|---|
| | | Retrieved | Not retrieved |
| **Reality** | Relevant | $TP$ | $FN$ |
| | Not relevant | $FP$ | $FN$ |

Table 3.2: Confusion matrix

Those values are then used to compute *recall* ($R$) and *precision* ($P$), two frequently used measures. The recall is the fraction of relevant characters that have been successfully retrieved, and the precision is the fraction of retrieved characters that are relevant to the query. Formally,

$$R = \frac{TP}{TP + FN} \ ,$$

$$P = \frac{TP}{TP + FP} \ .$$

Two types of thresholds can be used to be compute recall and precision: if we keep the same threshold for all keywords, it is a *global* threshold; if the performance is measured over each keyword and then averaged, we talk about *local* thresholds. The metric used to evaluate global thresholds is the *average precision* ($AP$): the associated recall and precision values for all threshold values are plotted to form the recall-precision curve, the $AP$ is simply the area under this curve. Global thresholds are evaluated with the *mean average precision* ($MAP$), the mean of each keyword's $AP$. All the values and metrics are computed using the `trec_eval`[3] software.

[3] `https://trec.nist.gov/trec_eval/`

# Chapter 4

# Experiments

This chapter will cover the experiments conducted in this work: comparing `Contour` and `Keypoint` graphs, validating their parameters as well as the cost function parameters, and finally discussing the keyword spotting experiences and their results, both on GW and on the papyri.

## 4.1 Comparison of graph representations

A first comparison of the two representations was made on the word images of the George Washington dataset. The baseline graphs are the `Keypoint` graphs created in [1], the dataset can be found on the *Histograph* website[1]. The graphs have a distance threshold $D = 4$.

Our `Keypoint` implementation has distance thresholds $D = 2, 3, 4, 5, 6, 8, 10, 12$, the `Contour` graphs without Douglas-Peucker also have distance thresholds $D = 2, 3, 4, 5, 6, 8, 10, 12$. To use the Douglas-Peucker algorithm, we kept all nodes, $D = 1$, and took approximation values $\varepsilon = 0.5, 1, 2, 4$.

The Table 4.1 shows the generation time and the average node count for the 4893 word images in the GW dataset for all graph representations.

Two elements can be extracted from these results: first, our `Keypoint` graphs have fewer nodes than the baseline, the reason must lie in the implementation, in the connected subcomponents creation or in the equidistant points selection for example. The second element is the duration difference: the `Keypoint` graphs are 50 to 70 times slower to generate than `Contour`. The problem most probably lies in the code, we have to search through the nodes and the edges many times and the code has not been optimized at all.

The time needed to generate bigger graphs can be found in Figure 4.1: the time needed to extract graphs from seven increasingly bigger patches of the same binarized document with one `Keypoint` ($D = 4$) and one `Contour` ($D = 8$) representation. We can see clearly that the `Contour` algorithm scales poorly with the size of the graph! The biggest image tested here was only approximately 1000 by 1000 pixels, a fifth of the original document.

We ran the experience as described in [1]. We used the same validation set : ten keywords, each appearing at least ten times with a maximum of 900 additional random words for a total of 1000 words. We also kept the optimal cost function parameters for `Keypoint`: $\tau_v = 4$, $\tau_e = 1$, $\alpha = 0.5$, $\beta = 0.1$. The results are evaluated on the *mean average precision* ($MAP$), Table 4.2 shows the results with the corresponding runtimes. The baseline $MAP$, the best `Keypoint` and best `Contour` results are shown in bold.

There is no sensible difference in runtime between different representations with the same number of nodes, but the differences in MAP are more interesting: our implementation of the `Keypoint` graphs is over 10% worse than the baseline, which indicates that there may be a problem with the implementation. Both versions of `Contour` graphs outperform our `Keypoint`, the version simplified with Douglas-Peucker even comes within 2%

[1] Histograph, `www.histograph.ch`

| | Keypoint | | Contour | |
|---|---|---|---|---|
| | Generation | Mean Nodes | Generation | Mean Nodes |
| $D = 4$ (Baseline) | | 86 | | |
| $D = 2$ | 1049 s | 140 | 36.5 s | 297 |
| $D = 3$ | 869 s | 99 | 26.7 s | 199 |
| $D = 4$ | 1278 s | 79 | 21.6 s | 150 |
| $D = 5$ | 1047 s | 67 | 18.9 s | 121 |
| $D = 6$ | 1267 s | 59 | 17.8 s | 102 |
| $D = 8$ | 1384 s | 49 | 15.7 s | 77 |
| $D = 10$ | 1346 s | 43 | 15.6 s | 63 |
| $D = 12$ | 1290 s | 39 | 13.6 s | 53 |
| $D = 1,\ \varepsilon = 0.5$ | | | 41.4 s | 182 |
| $D = 1,\ \varepsilon = 1$ | | | 20.3 s | 90 |
| $D = 1,\ \varepsilon = 2$ | | | 19.9 s | 58 |
| $D = 1,\ \varepsilon = 4$ | | | 17.6 s | 37 |

Table 4.1: Comparison of word graphs generation on GW

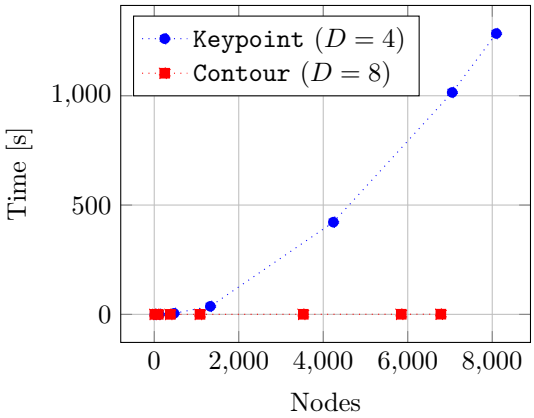



Figure 4.1: Generation time for bigger graphs

| | Keypoint | | Contour | |
|---|---|---|---|---|
| | Runtime | MAP | Runtime | MAP |
| $D = 4$ (Baseline) | 92.5 s | **0.798** | | |
| $D = 2$ | 369 s | 0.584 | 2059 s | 0.618 |
| $D = 3$ | 157 s | 0.645 | 770 s | 0.639 |
| $D = 4$ | 79.9 s | 0.678 | 493 s | 0.661 |
| $D = 5$ | 59.7 s | **0.690** | 301 s | 0.665 |
| $D = 6$ | 50.0 s | 0.658 | 175 s | 0.695 |
| $D = 8$ | 28.0 s | 0.650 | 73.4 s | 0.709 |
| $D = 10$ | 22.2 s | 0.658 | 68.0 s | **0.720** |
| $D = 12$ | 18.9 s | 0.620 | 42.3 s | 0.694 |
| $D = 1,\ \varepsilon = 0.5$ | | | 684 s | 0.674 |
| $D = 1,\ \varepsilon = 1$ | | | 54.0 s | **0.775** |
| $D = 1,\ \varepsilon = 2$ | | | 29.4 s | 0.729 |
| $D = 1,\ \varepsilon = 4$ | | | 22.3 s | 0.699 |

Table 4.2: Evaluation of graph representations on GW

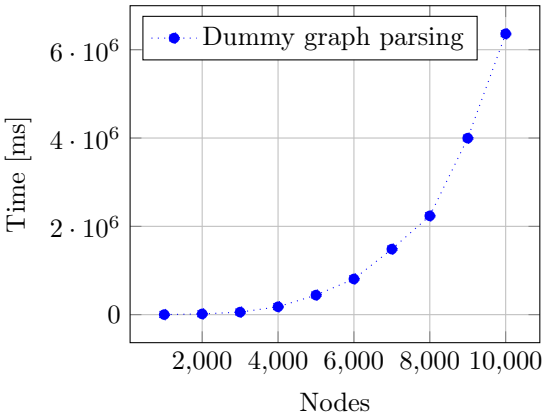



Figure 4.2: Parsing time evolution for dummy graphs

of the baseline. Interestingly, the best performing `Contour` representation ($D = 1, \varepsilon = 1$) also has a number of nodes close to the baseline.

Due to the slow generation of `Keypoint` graphs and their overall worse results, and due to time constraints, we decided to focus only on `Contour` graphs with Douglas-Peucker for the rest of the experiences.

## 4.2 Graph representation validation

Once we tested our system on bigger images – and thus bigger graphs – , we quickly realized the graph parsing also scaled poorly time wise. We do not use the full complexity of the Graph class, its implementation was not optimized for the size of the graphs we use. To illustrate this, we created dummy graphs, each ranging from 1000 to 10000 nodes and edges; the time taken to parse them can be found in Figure 4.2.

We decided to keep a maximum of 3000 to 4000 nodes in the graphs to ensure reasonable parsing times. We limited the sizes in the two following ways: first, by visually choosing high Douglas-Peucker approximation values $\varepsilon$ such that the graphs are still dense enough to preserve the characters' important features; and second, because the graphs were still too big, by cropping the original documents. We kept the top third of the GW pages, the papyri have been arbitrarily cropped to reach the node threshold. In [1], Stauffer *et al.* also limit the size of graphs to trade some accuracy for a better runtime.

A confirmation that the size of graphs had to be limited appeared later at the graph matching step: too many variables, values and results and results had to be stored at the same time, overfilling the heap space of the Java Virtual Machine and creating errors.

We finally kept the same representation on all documents: `Contour` graphs with Douglas-Peuckler approximation value $\varepsilon = 4$ for all documents: GW, the original papyri and the two reconstructions.

## 4.3 Cost function parameters validation

We can now optimise the cost function on the chosen graph representations. 81 parameters combinations are evaluated on each of the representations. We took three different values for each of the four parameters: $\tau_v, \tau_e \in \{4, 8, 16\}$; $\alpha, \beta \in \{0.3, 0.5, 0.7\}$. For time reasons, we did not keep 5 values per parameter as in [1], this reduced the number of different combinations from $5^4 = 625$ to $3^4 = 81$. On GW, we matched the word *the* on a cropped training page; on the three papyrus documents, we matched a character $\epsilon$ on the patches.

The best parameter combinations for each combination are shown in Table 4.3.

| `Contour` ($D = 1, \varepsilon = 4$) | Parameters | | | |
|---|---|---|---|---|
| | $\tau_v$ | $\tau_e$ | $\alpha$ | $\beta$ |
| Original Papyrus | 8 | 4 | 0.7 | 0.3 |
| Reconstructed Papyrus | 16 | 4 | 0.7 | 0.3 |
| Reconstructed Bold Papyrus | 16 | 4 | 0.7 | 0.3 |
| George Washington | 16 | 4 | 0.7 | 0.3 |

Table 4.3: Optimal cost function parameters

| Word | Occurrences | Characters |
|---|---|---|
| *de* | 1 | 2 |
| *etc.* | 5 | 4 |
| *made* | 2 | 4 |
| *Letters* | 5 | 6 |
| *Orders* | 5 | 6 |
| *remain* | 2 | 6 |
| *escort* | 2 | 6 |
| *Virginia* | 2 | 9 |
| *Alexandria* | 2 | 10 |
| *Instructions.* | 3 | 13 |

Table 4.4: GW: Keyword occurrences in cropped pages test set

## 4.4 Results on GW

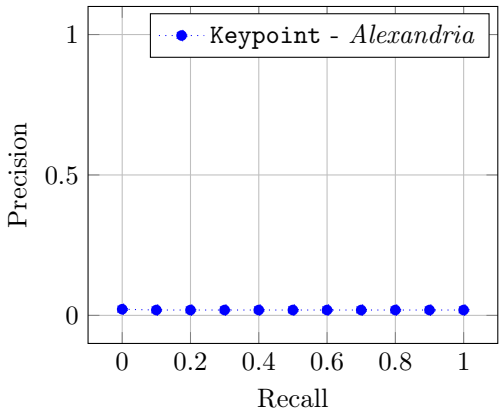
The GW dataset has a four-fold cross-validation split. We only used the first fold as the split for our experience: the test set includes five pages. As explained in section 4.2, the pages all have been cropped to their top third. We updated the list of the fold's query words to only keep the keywords appearing in the cropped pages, sorted them by apparition number and kept ten words of different lengths. The keywords with their character count and occurrences are listed in Table 4.4. Each word has to to matched separately, since matching all of them together would overflow the memory of the machine. This means that the target document graphs had to be parsed again for each word, adding to the time needed for the experience.

The $MAP$ resulting from each word query on GW can be found in Table 4.5 with their mean highlighted in bold, the corresponding recall-precision curves are shown in Figure 4.3. The results did not look promising: only one word crossed the bar of an $MAP$ of 20 points and three are over 10%; the recall-precision curves also reflect this fact: the worst have very low precision even for zero or small recall, possibly indicating a bad model.

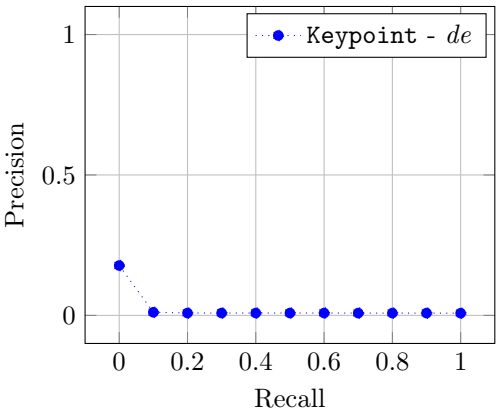
The Figures 4.4 and 4.5 show the best matches found for two words (*Orders* has the best $MAP$ while

| Word | $MAP$ |
|---|---|
| *Alexandria* | 0.0174 |
| *de* | 0.0127 |
| *escort* | 0.0127 |
| *etc.* | 0.0386 |
| *Instructions.* | 0.0816 |
| *Letters* | 0.1745 |
| *made* | 0.1096 |
| *Orders* | 0.2202 |
| *remain* | 0.0407 |
| *Virginia* | 0.0426 |
| Mean | **0.0771** |

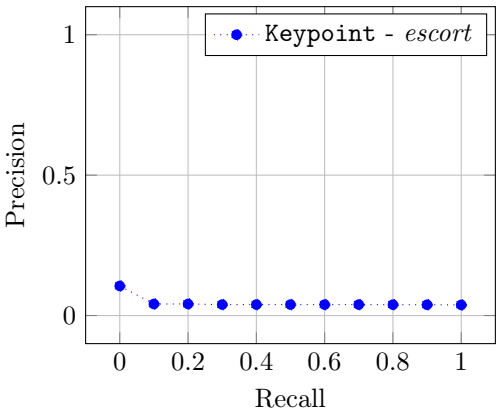
Table 4.5: $MAP$ for each query word on GW



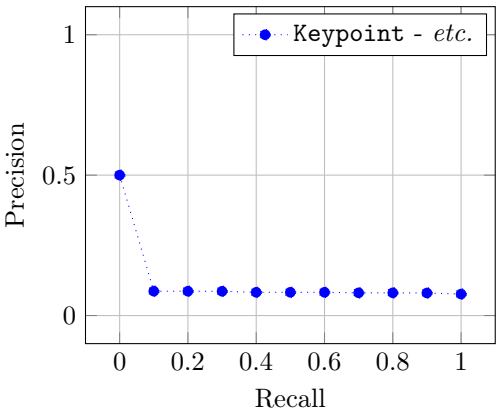

4.3.1: Local threshold: *Alexandria*




4.3.2: Local threshold: *de*

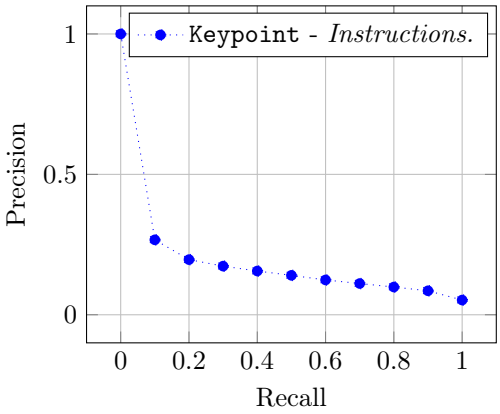



4.3.3: Local threshold: *escort*

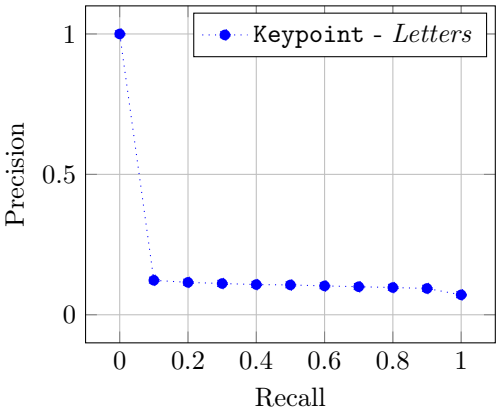



4.3.4: Local threshold: *etc.*

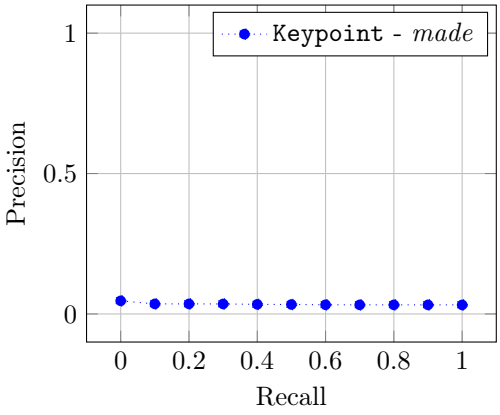
Figure 4.3: Recall-precision curves ($MAP$) on GW, all keywords



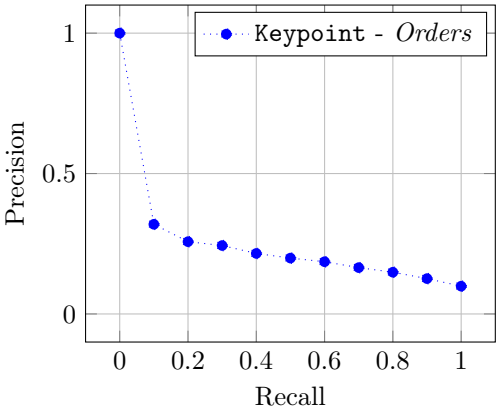

4.3.5: Local threshold: *Instructions.*



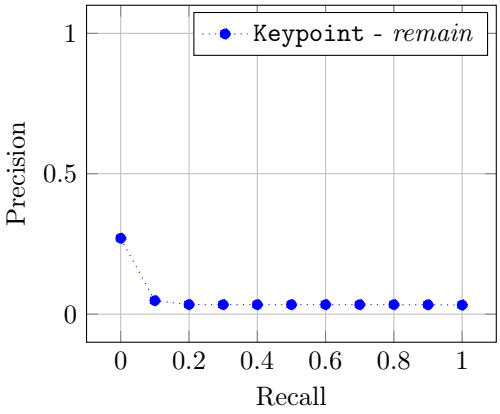

4.3.6: Local threshold: *Letters*



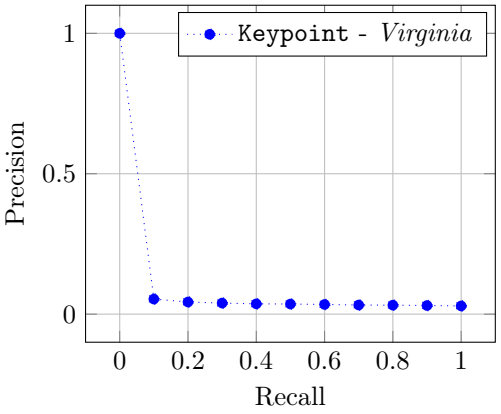

4.3.7: Local threshold: *made*




4.3.8: Local threshold: *Orders*




4.3.9: Local threshold: *remain*




4.3.10: Local threshold: *Virginia*

Figure 4.3: Recall-precision curves ($MAP$) on GW, all keywords

*Alexandria* almost has the worst result) each on a page with an occurrence of the word, labelled in decreasing distance order. Bounding boxes overlapping better matches have not been selected, only the ten best distinct ones appear on the figures. Tables 4.6 and 4.7 show the distances of the matches to the source word, with the correct matches outlined in bold.

The interesting points that can be taken from Figure 4.4 are as follows: the one occurence appearing on the page has been recognized by the model, however with a one-letter shift. The other attempted matches are mostly on other long words with high reaching letters that the program tries to match with the *A*, *l* or *d*. The matches produced with *Orders* on Figure 4.5 show some success: the word's occurence is correctly matched, the other matches are well aligned with the text lines.

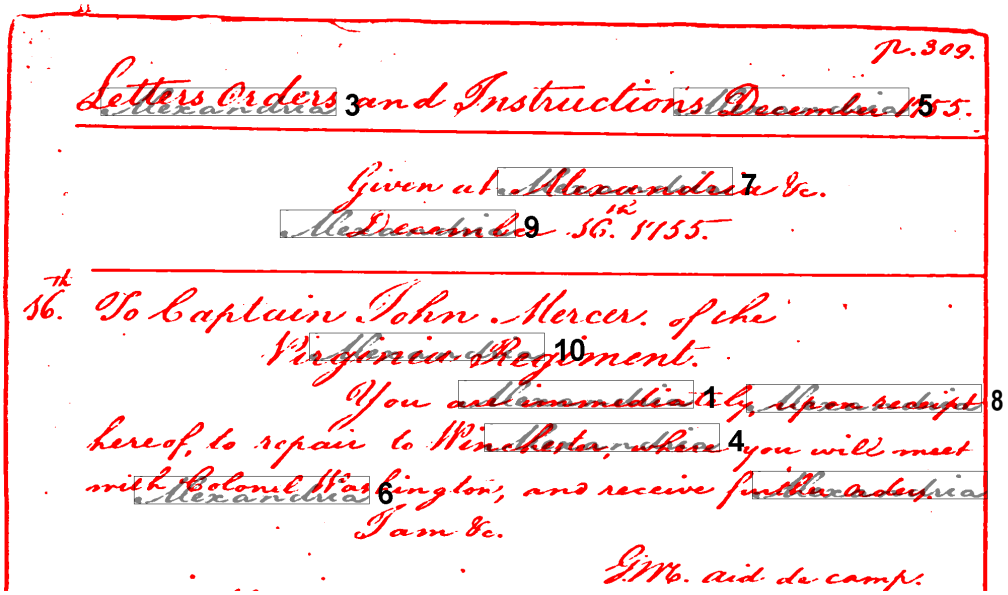



Figure 4.4: Best matches for *Alexandria* on a selected GW page

| Match no. | Distance |
|---|---|
| 1 | 278.84 |
| 2 | 294.98 |
| 3 | 298.72 |
| 4 | 301.40 |
| 5 | 307.24 |
| 6 | 307.67 |
| **7** | **309.83** |
| 8 | 311.18 |
| 9 | 313.60 |
| 10 | 323.20 |

Table 4.6: Distances of the 10 best matches for *Alexandria* on a GW page

We can now reflect on these results: from the low *MAP* and the recall-precision curves, we see that the adaptation of the base framework from [1] to segmentation-free KWS does not produce such good results.

There are different possible reasons for this, some of them come from the increase in complexity - and thus of the time needed to run the queries. To ensure a reasonable project run time and not overflow the Java Virtual Machine's memory, we had to restrict several options. As mentioned in Subsection 4.1, our `Keypoint` implementation was not performing good enough to allow us to use it for document graph generation, we therefore only kept one graph representation. Similarly, the graph parsing time had to be limited, we set a maximum amount of nodes in the document graphs and enforced it by cropping the documents and taking unoptimal Douglas-Peucker approximation values. The cost function parameters combinations were also reduced by a factor of almost eighty. The size of the windows could also be changed: one could test a range of window sizes, *e.g.* from half to twice the dimension of the query keyword to account for the differences in the writing. An even better possibility would also be slightly vary the window's proportions. In this work, we only parsed

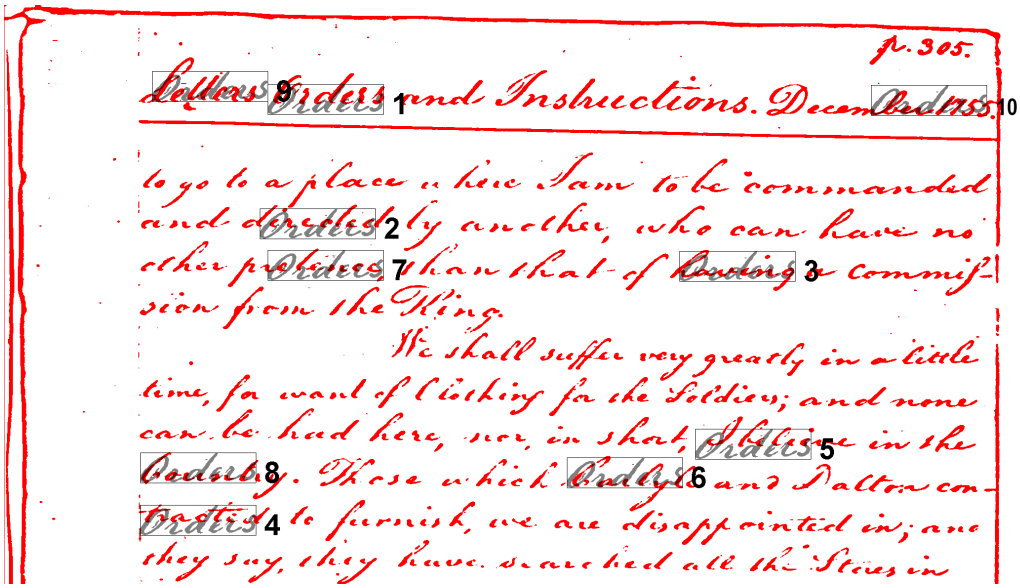



Figure 4.5: Best matches for *Orders* on a selected GW page

| Match no. | Distance |
|---|---|
| **1** | **161.72** |
| 2 | 186.16 |
| 3 | 187.33 |
| 4 | 191.80 |
| 5 | 192.95 |
| 6 | 193.35 |
| 7 | 194.44 |
| 8 | 195.07 |
| 9 | 196.66 |
| 10 | 196.82 |

Table 4.7: Distances of the 10 best matches for *Orders* on a GW page

the document with the original window size.

We also only used the first fold of the cross-validation split and small amount of query keywords. The groundtruth as well was not ideal for this task as mentioned in Section 3.4, it would have been better to have the word positions on the whole page.

All these stacked steps may have too much hindered the model's capacities for the complexity of Segmentation-free Keyword Spotting. This reduction is especially visible on GW, a document on which good results were obtained with segmented models.

## 4.5 Results on Papyri

As for the GW pages, only patches were kept for the papyri. We decided to keep a unique papyrus, choosing the document with the best looking characters: it is the document called *P.Corn. Inv. MSS. A 101. XIII.* The others documents were too difficult to analyze, the manual recognition and labelling of the characters were already hard to achieve. We compared the original binarization and the two manual reconstructions by searching for an $\epsilon$ character. The papyrus needed three runs, one for each of the versions.

The Table 4.8 shows the $MAP$ for the three documents. The first result that can be seen is the apparent hierarchy between the versions: the bold reconstruction performs better than the thin reconstruction by 8 points, which in turn outperforms the original document by another 11 points. This behavior also appears on

| Document | $MAP$ |
|---|---|
| Original Papyrus | 0.2205 |
| Reconstructed Papyrus | 0.3331 |
| Reconstructed Bold Papyrus | 0.4181 |

Table 4.8: $MAP$ for the papyrus versions

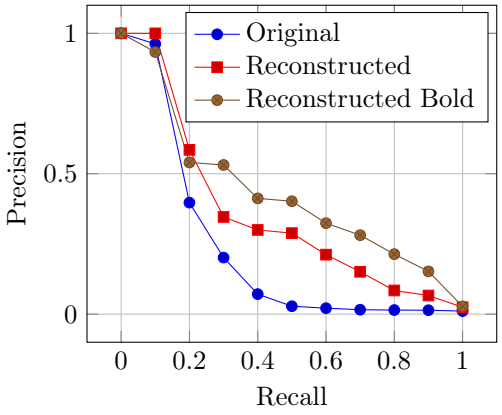



Figure 4.6: Recall-precision curves ($MAP$) on Papyrus, character $\epsilon$

the recall-precision curves shown in Figure 4.6: the curves are superimposed only on the extreme recall values and are clearly ordered on the other values.

The clear improvement between the reconstructions and the original document was expected with the noise removal and the already clean contours of the characters. The double contours of the bold reconstruction may explain its surprising better performance.

The Figures 4.7 to 4.9 illustrate the ten best distinct matches on each document. The matched $\epsilon$ is displayed in grey in its bounding box over the red colorized document. The Tables 4.9, 4.10 and 4.11 show the distances of these matches to the source character, the bold lines indicate the correct matches. On the Original Papyrus (Figure 4.7), only two of the overall six $\epsilon$ appear in the top ten. Only one character was missed in Figure 4.8 while all six were successfully matched in Figure 4.9, underlining again the better results of the reconstructions. The wrong matches are an interesting consideration: on the reconstructions, they often happen on crescent shaped characters such as $\varsigma$, $o$ or $\phi$; whereas this is not the case on the original document: the wrong matches appear to be between characters or on all kinds of shapes. This further indicates how hard the use of the original document's graphs seems to be.

| Match no. | Distance |
|---|---|
| **1** | **30.30** |
| 2 | 105.82 |
| 3 | 108.73 |
| 4 | 108.93 |
| 5 | 110.33 |
| 6 | 110.57 |
| 7 | 110.59 |
| **8** | **110.72** |
| 9 | 110.78 |
| 10 | 112.0 |

Table 4.9: Distances of the 10 best matches on original papyrus

The differences between the three Papyri documents and the improvement on the reconstructions lead to interesting conclusions: the original papyri are a really hard dataset, but we have some confirmation that on manually cleaned documents, the model performs relatively good. We can only hope now that the difficulty of the dataset does not prove itself to be insurmountable.

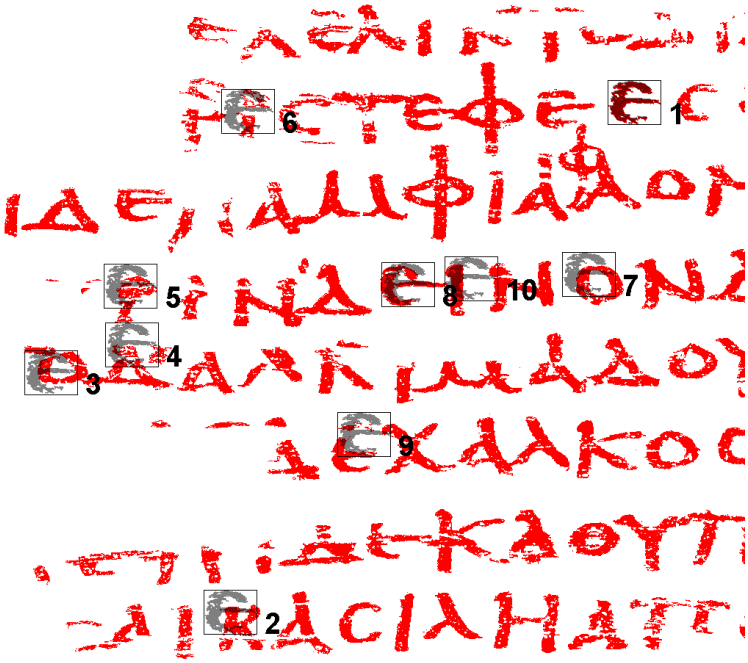



Figure 4.7: Best matches on original papyrus

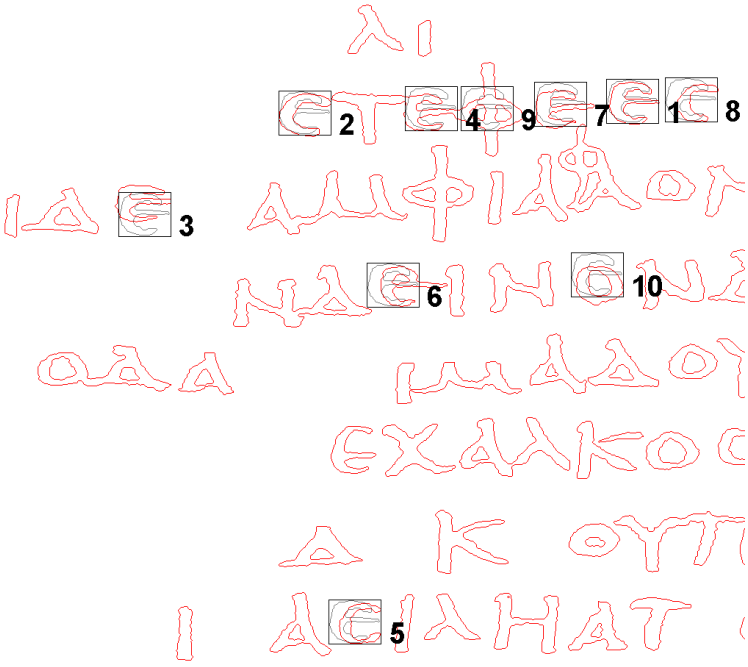



Figure 4.8: Best matches on reconstructed papyrus

| Match no. | Distance |
|---|---|
| **1** | **4.20** |
| 2 | 69.21 |
| **3** | **71.29** |
| 4 | 72.68 |
| **5** | **77.09** |
| **6** | **77.23** |
| **7** | **78.89** |
| 8 | 81.03 |
| 9 | 82.36 |
| 10 | 84.49 |

Table 4.10: Distances of the 10 best matches on reconstructed papyrus

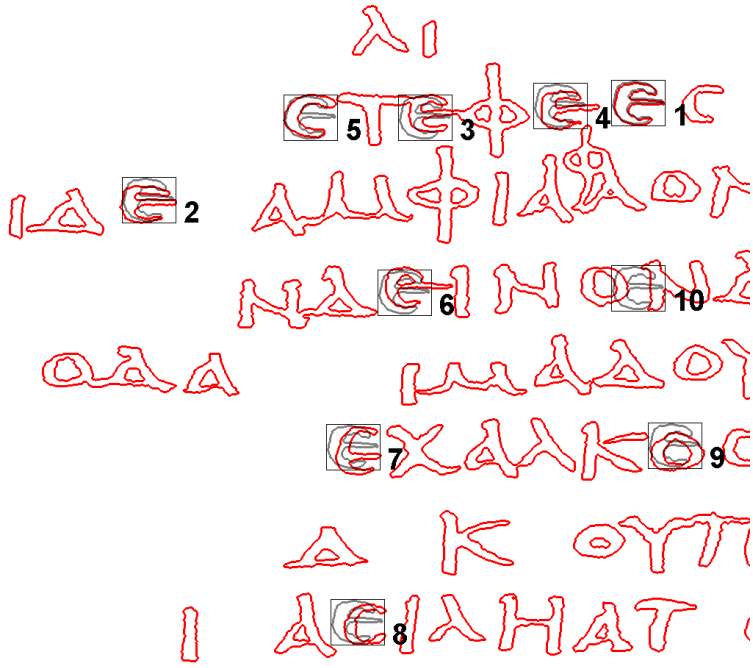



Figure 4.9: Best matches on bold reconstructed papyrus

| Match no. | Distance |
|---|---|
| **1** | **7.20** |
| **2** | **60.64** |
| **3** | **62.63** |
| **4** | **66.74** |
| 5 | 68.70 |
| **6** | **70.33** |
| **7** | **75.91** |
| 8 | 77.10 |
| 9 | 78.40 |
| 10 | 85.68 |

Table 4.11: Distances of the 10 best matches on bold reconstructed papyrus

# Chapter 5

# Conclusion

Digitization is a way to help preserve historical documents. However, due to the variations in handwriting, the diverse degradations and often smaller sample size of documents, getting a complete transcription generally becomes really challenging. Keyword Spotting aims to alleviate this problem by directly retrieving all occurrences of a query keyword in the document. Stauffer *et al.* propose a graph-based KWS framework in [1].

In order to use this framework on the *DIBCO Papyrus dataset*, a set of *Iliad* papyri, we adapt it for segmentation-free KWS: smaller graphs are extracted from the document and are then matched with the query keyword. We also introduce `Contour` graphs, a new graph representation that takes the stroke into account.

A first test is run on the *George Washington* dataset, it shows that the step to segmentation-free induces a huge increase in complexity, forcing us to limit the options and scope of our experiments. The results obtained are objectively worse than those from [1], possibly because of the limitations introduced: *e.g.* a maximum graph size, a reduced parameter range and reduced training samples.

We then test the framework on the papyri and can extract the following: performing KWS on such documents is intrinsically more difficult, because of the quality of the papyrus itself: holes or darker spots – in opposition to light paper – complicate the correct outlining of the characters and thus produce worse graphs. However, on hand-reconstructed binarizations, we see a clear result improvement, a promising result for this segmentation-free framework.

What are the possible next steps for this framework? As mentioned above, a certain number of choices have been made to reduce the runtime of our program; more time or better infrastructure would allow to test a wider range of cost function parameters, generate graphs with a higher number of nodes, match more keywords and characters on uncropped versions of all documents for example. The size as well as the proportions of the windows used for the graph extraction could also vary in accounting for the possible variation in handwriting. The *Graph* class could be reimplemented, it currently has more features than we need and is not really optimised.

Finally, more graph representations could be tested. `Keypoint` graphs were the best performing representation in [1] but our implementation seems to be faulty and takes too much time generating the graphs, it was therefore left out of our experiments.

# Appendix A

# Figures

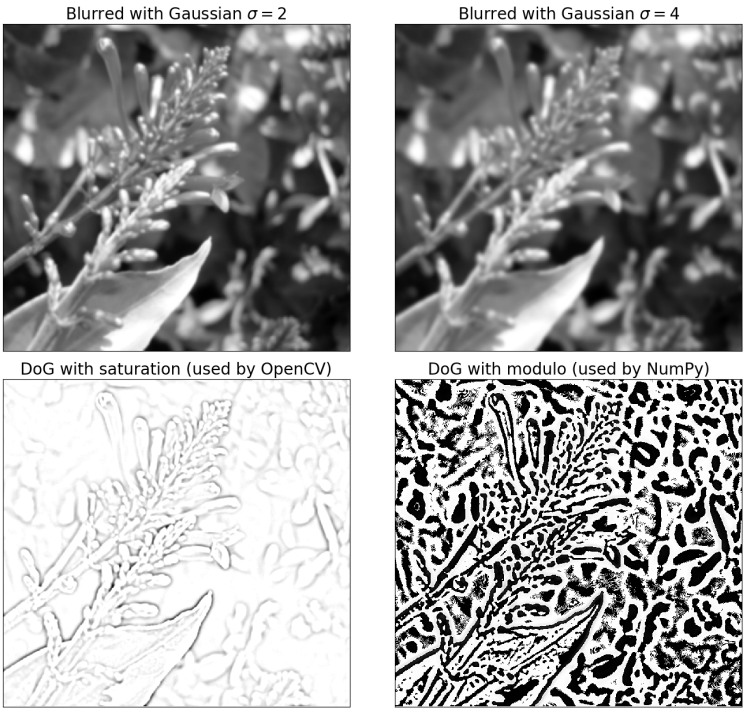
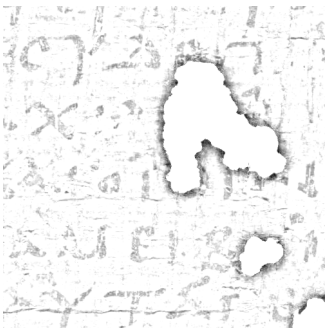
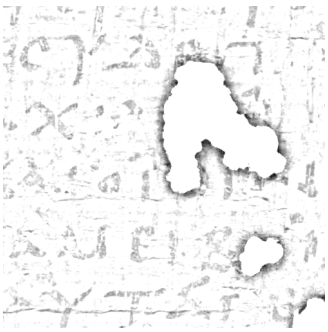



Figure A.1: Comparison of both substraction methods for the DoG: two blurred images (top) and their substraction using OpenCV's saturated method (bottom left) and NumPy's modulo method (bottom right) [1]

[1] Source of the picture: https://upload.wikimedia.org/wikipedia/commons/d/da/Flowers_before_difference_of_gaussians.jpg

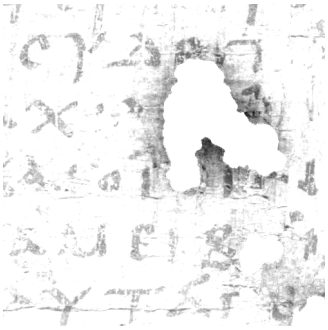


A.2.1: DoG: $\sigma_1 = 0.1, \sigma_2 = 10$

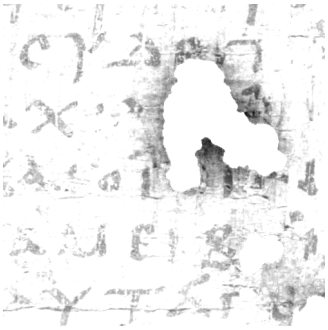


A.2.2: DoG: $\sigma_1 = 0.5, \sigma_2 = 10$

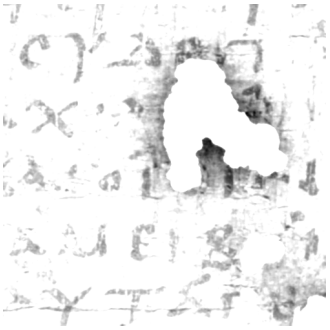
A.2.3: DoG: $\sigma_1 = 1, \sigma_2 = 10$



A.2.4: DoG: $\sigma_1 = 0.1, \sigma_2 = 40$

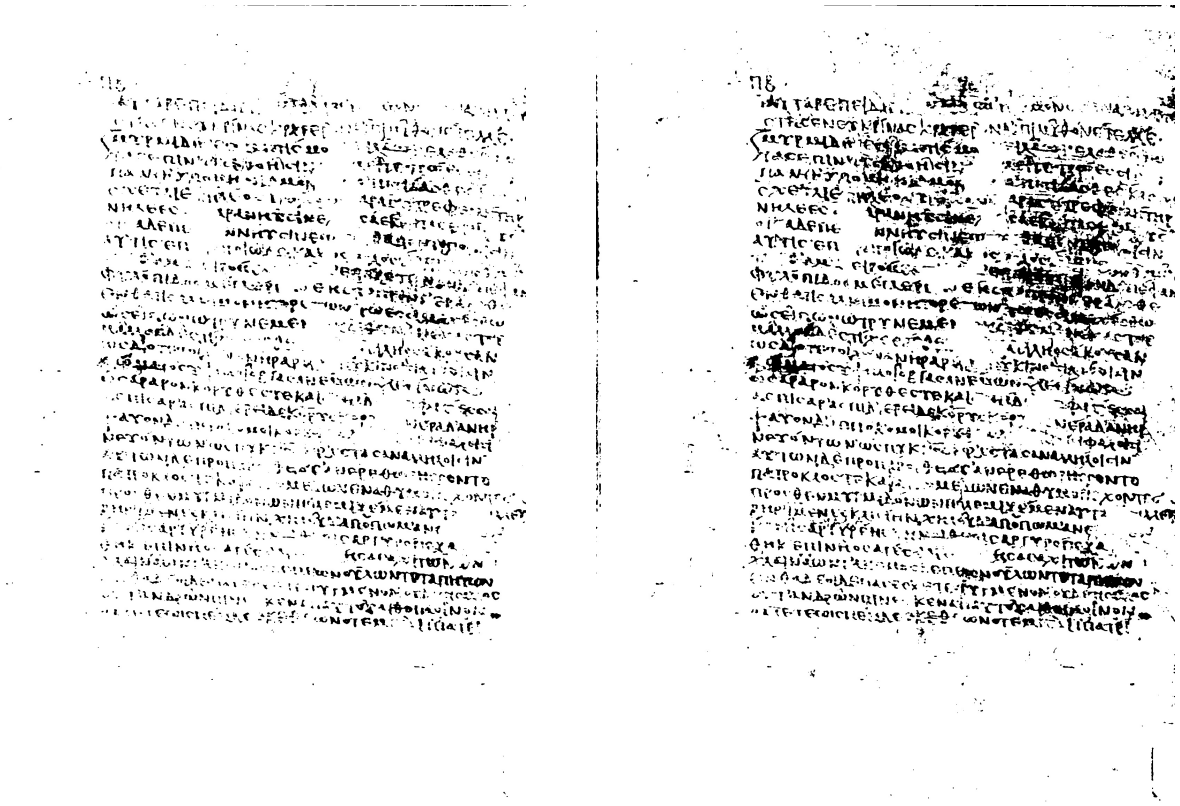


A.2.5: DoG: $\sigma_1 = 0.5, \sigma_2 = 40$



A.2.6: DoG: $\sigma_1 = 1, \sigma_2 = 40$

Figure A.2: Comparison of the different parameters for the Difference of Gaussians



A.3.1: Median Blur of size 5, Binarization with Threshold 70

A.3.2: Median Blur of size 5, Binarization with Threshold 100

Figure A.3: Comparison of different thresholds for binarization

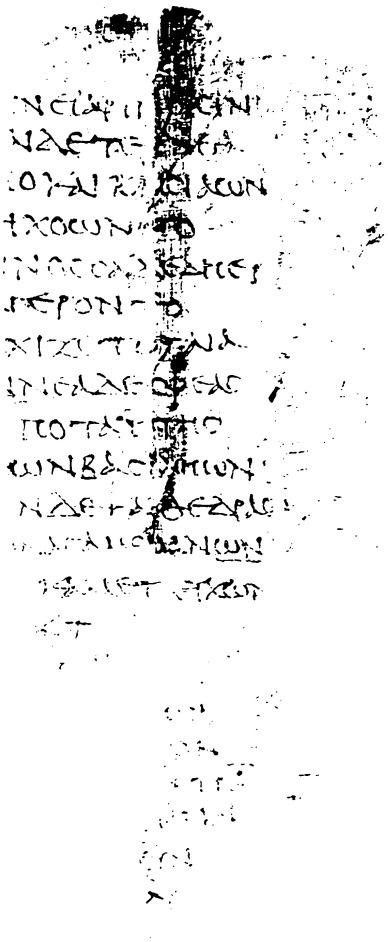
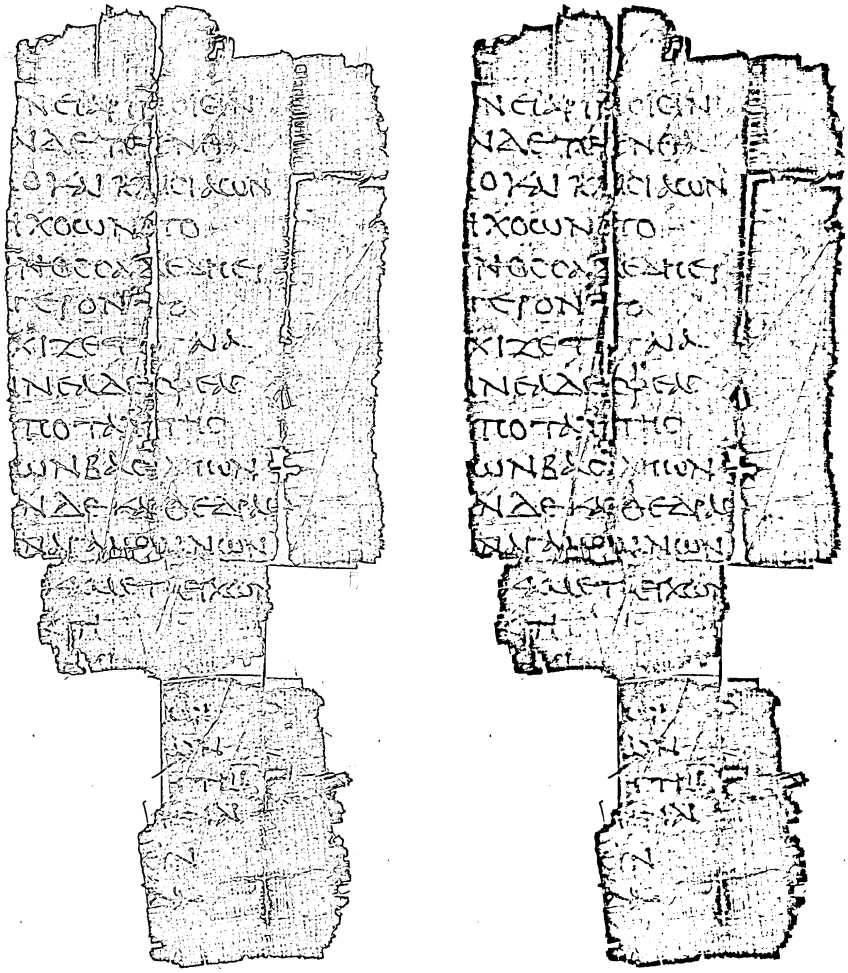


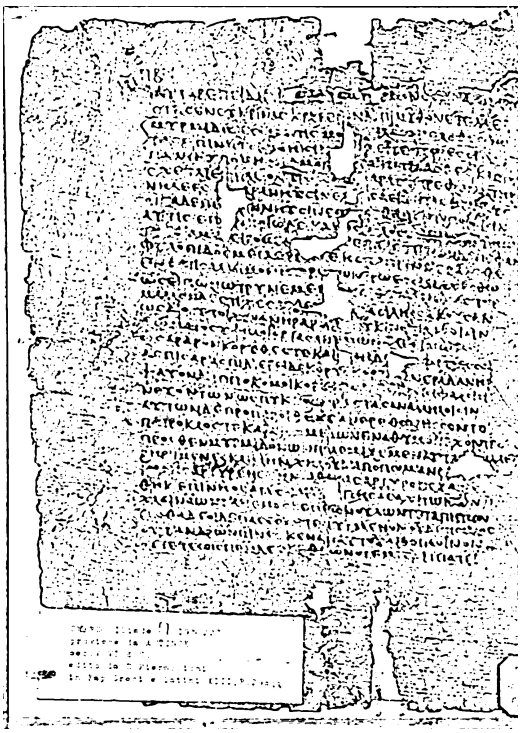
Figure A.4: Median Blur of size 5, binarization with threshold 80



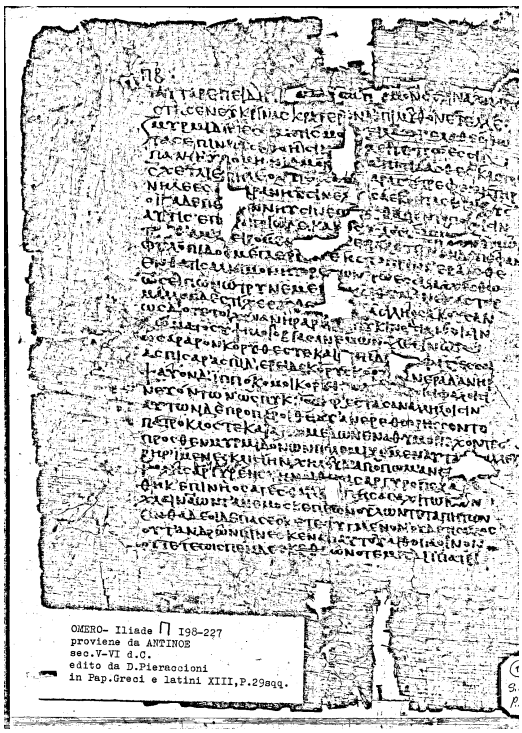
A.5.1: Median Blur of size 7, Adaptive Gaussian Thresholding

A.5.2: Binarization with threshold 250, $\sigma_1 = 0.5, \sigma_2 = 10$

Figure A.5: Comparison of different binarization results



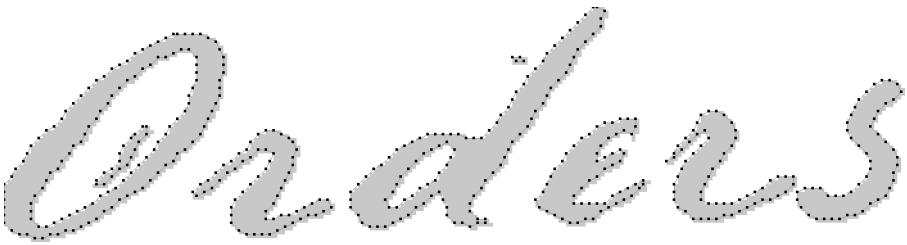
A.6.1: Median Blur of size 5, Adaptive Mean Thresholding



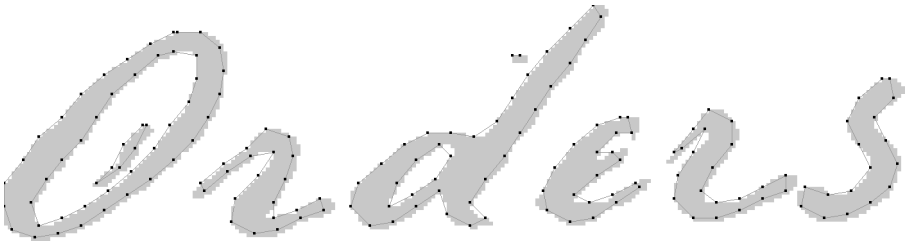

A.6.2: Binarization with threshold 250, $\sigma_1 = 0.5, \sigma_2 = 10$

Figure A.6: Comparison of different binarization results

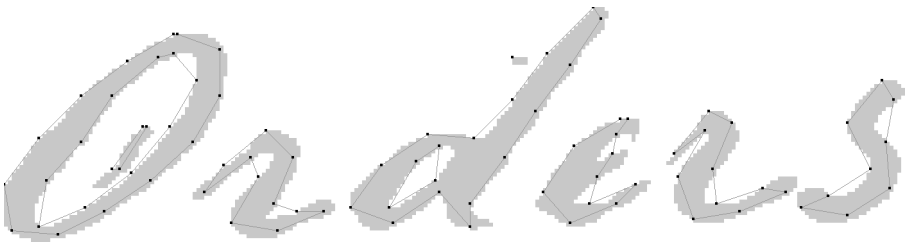
Figure A.7: Binarized word of GW



A.8.1: `Contour` graph, $D = 2$

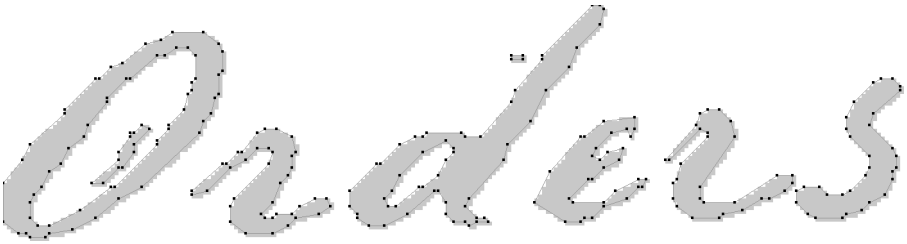


A.8.2: `Contour` graph, $D = 6$

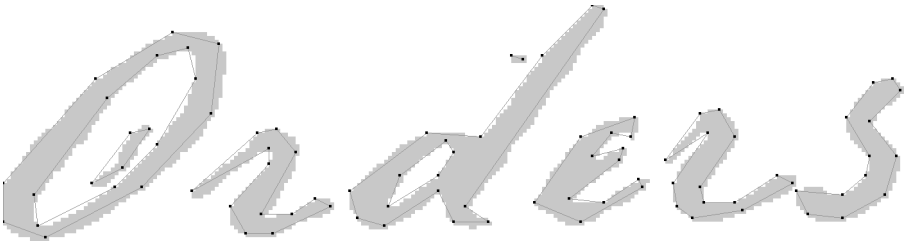


A.8.3: `Contour` graph, $D = 12$

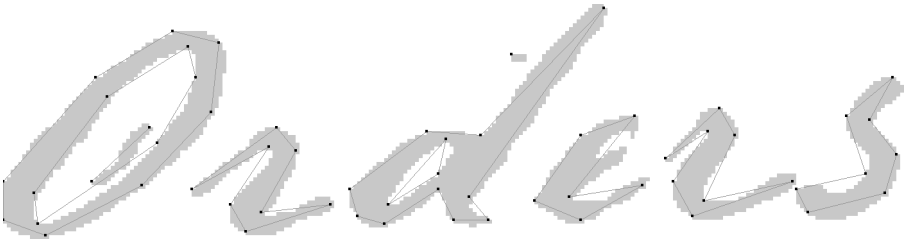
Figure A.8: `Contour` graphs on *Orders*, without Douglas-Peucker



A.9.1: `Contour` graph, $D = 1, \varepsilon = 0.5$



A.9.2: `Contour` graph, $D = 1, \varepsilon = 2$

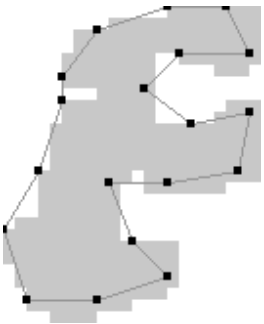


A.9.3: `Contour` graph, $D = 1, \varepsilon = 4$

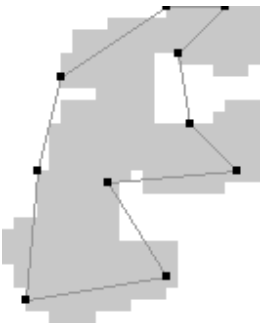
Figure A.9: `Contour` graphs on *Orders*, with Douglas-Peucker

Figure A.10: Small binarized character of papyri

A.11.1: `Contour` graph, $D = 2$

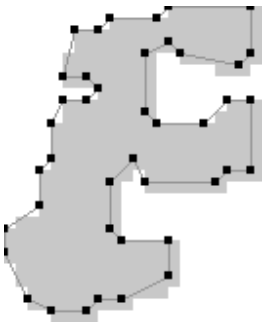


A.11.2: `Contour` graph, $D = 6$

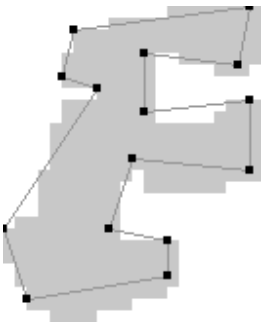


A.11.3: `Contour` graph, $D = 12$

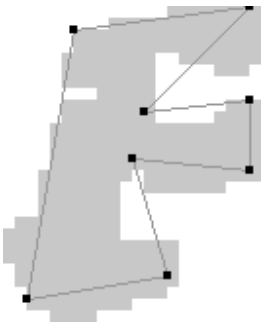
Figure A.11: `Contour` graphs on a small $\epsilon$, without Douglas-Peucker



A.12.1: `Contour` graph, $D = 1, \varepsilon = 0.5$



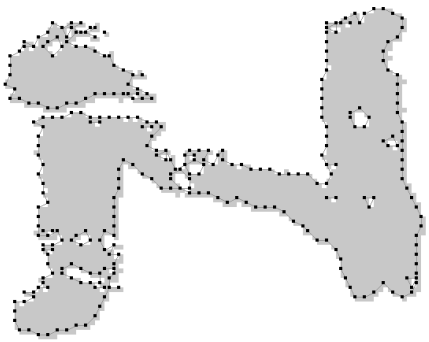
A.12.2: `Contour` graph, $D = 1, \varepsilon = 2$



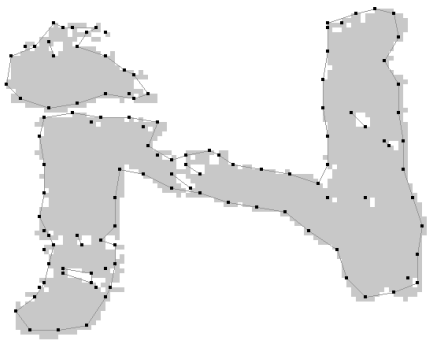
A.12.3: `Contour` graph, $D = 1, \varepsilon = 4$

Figure A.12: `Contour` graphs on a small $\epsilon$, with Douglas-Peucker

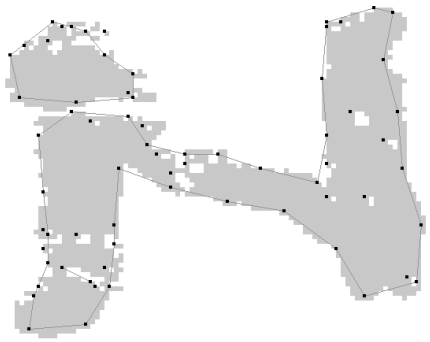
Figure A.13: Binarized character of papyri



A.14.1: `Contour` graph, $D = 2$

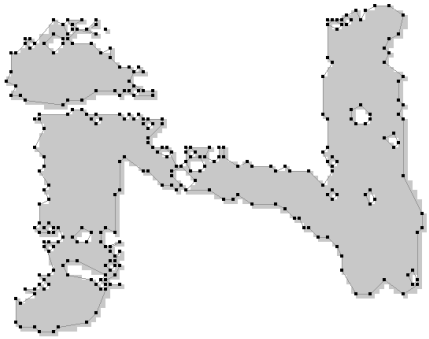


A.14.2: `Contour` graph, $D = 6$

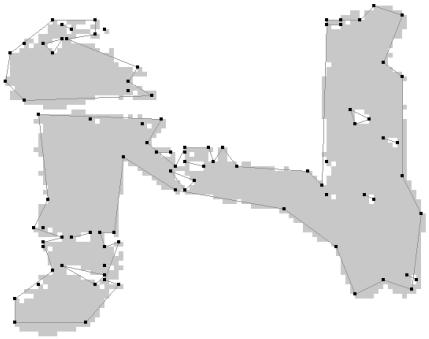


A.14.3: `Contour` graph, $D = 12$

Figure A.14: `Contour` graphs on a $N$, without Douglas-Peucker



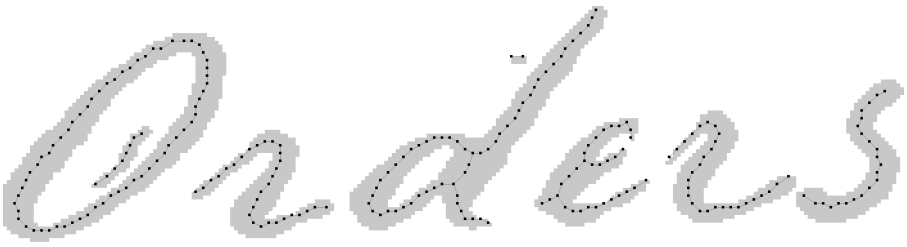
A.15.1: `Contour` graph, $D = 1, \varepsilon = 0.5$



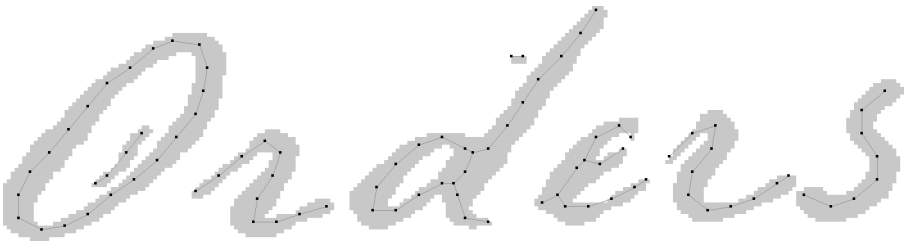
A.15.2: `Contour` graph, $D = 1, \varepsilon = 2$

A.15.3: `Contour` graph, $D = 1, \varepsilon = 4$

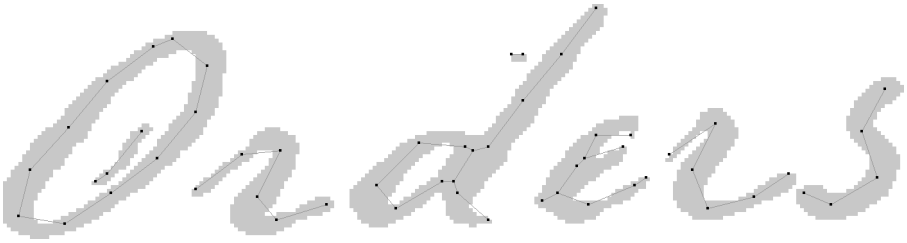
Figure A.15: `Contour` graphs on a $N$, with Douglas-Peucker



A.16.1: `Keypoint` graph, $D = 2$

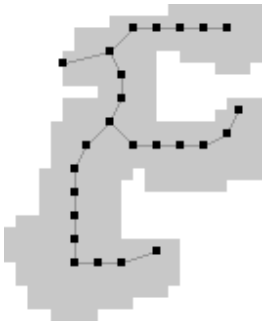


A.16.2: `Keypoint` graph, $D = 6$

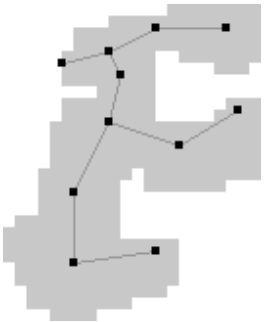


A.16.3: `Keypoint` graph, $D = 12$

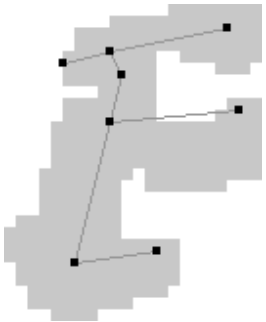
Figure A.16: `Keypoint` graphs on *Orders*



A.17.1: `Keypoint` graph, $D = 2$



A.17.2: `Keypoint` graph, $D = 6$



A.17.3: `Keypoint` graph, $D = 12$

Figure A.17: `Keypoint` graphs on a small $\epsilon$

A.18.1: `Keypoint` graph, $D = 2$

A.18.2: `Keypoint` graph, $D = 6$

A.18.3: `Keypoint` graph, $D = 12$

Figure A.18: `Keypoint` graphs on a $N$

# Bibliography


[1] M. Stauffer, A. Fischer, and K. Riesen, *Graph-Based Keyword Spotting*. World Scientific, 2019.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[3] D. Lowe, "Object Recognition From Local Scale-Invariant Features," in *Proc. Int. Conf. on Computer Vision (ICCV)*, vol. 2, pp. 1150–1157 vol.2, 1999.

[4] D. Aldavert, M. Rusiñol, R. Toledo, and J. Lladós, "A Study of Bag-of-Visual-Words Representations for Handwritten Keyword Spotting," vol. 18, pp. 223–234, 2015.

[5] J. Almazán, A. Gordo, A. Fornés, and E. Valveny, "Word Spotting and Recognition with Embedded Attributes," in *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2014.

[6] A. Fischer, A. Keller, V. Frinken, and H. Bunke, "Lexicon-free Handwritten Word Spotting Using Character HMMs," *Pattern Recognition Letters (PRL)*, vol. 33, pp. 934–942, 2012.

[7] S. Sudholt and G. A. Fink, "PHOCNet : A Deep Convolutional Neural Network for Word Spotting in Handwritten Documents," in *Proc. Int. Conf. on Frontiers in Handwriting Recognition (ICFHR)*, 2016.

[8] S. Ghosh and E. Valveny, "R-PHOC: Segmentation-Free Word Spotting using CNN," 2017.

[9] L. Xing, Z. Tian, W. Huang, and M. R. Scott, "Convolutional Character Networks," in *Proc. Int. Conf. on Computer Vision (ICCV)*, 2019.

[10] N. R. Howe, "Inkball Models for Character Localization and Out-of-Vocabulary Word Spotting," in *Proc. Int. Conf. on Document Analysis and Recognition (ICDAR)*, pp. 381–385, IEEE Computer Society, 2015.

[11] I. Pratikakis, K. Zagoris, X. Karagiannis, L. Tsochatzidis, T. Mondal, and I. Marthot-Santaniello, "ICDAR 2019 Competition on Document Image Binarization (DIBCO 2019)," in *Int. Conf. on Document Analysis and Recognition (ICDAR)*, pp. 1547–1556, 2019.

[12] A. Fischer, K. Riesen, and H. Bunke, "Graph Similarity Features for HMM-Based Handwriting Recognition in Historical Documents," in *Proc. Int. Conf. on Frontiers in Handwriting Recognition (ICFHR)*, pp. 253–258, 2010.

[13] T. Y. Zhang and C. Y. Suen, "A Fast Parallel Algorithm for Thinning Digital Patterns," *Communications of the ACM*, vol. 27, pp. 236–239, Mar. 1984.

[14] H. Bunke and G. Allermann, "Inexact Graph Matching for Structural Pattern Recognition," *Pattern Recognition Letters (PRL)*, vol. 1, no. 4, pp. 245–253, 1983.

[15] A. Sanfeliu and K.-S. Fu, "A Distance Measure Between Attributed Relational Graphs for Pattern Recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 3, pp. 353–362, 1983.

[16] A. Fischer, C. Y. Suen, V. Frinken, K. Riesen, and H. Bunke, "Approximation of Graph Edit Distance Based on Hausdorff Matching," *Pattern Recognition*, vol. 48, no. 2, pp. 331 – 343, 2015.

[17] D. P. Huttenlocher, W. J. Rucklidge, and G. A. Klanderman, "Comparing Images Using the Hausdorff Distance Under Translation," in *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 654–656, 1992.