Noname manuscript No.
(will be inserted by the editor)

Survey and Empirical Comparison of Different Approaches for Text Extraction from Scholarly Figures

Falk Böschen · Tilman Beck · Ansgar Scherp

Received: date / Revised version: date

Abstract Different approaches have been proposed in the past to address the challenge of extracting text from scholarly figures. However, until recently, no comparative evaluation of the different approaches had been conducted. Thus, we performed an extensive study of the related work and evaluated in total 32 different approaches. In this work, we perform a more detailed comparison of the 7 most relevant approaches described in the literature and extend to 37 systematic linear combinations of methods for extracting text from scholarly figures. Our generic pipeline, consisting of six steps, allows us to freely combine the different possible methods and perform a fair comparison. Overall, we have evaluated 44 different linear pipeline configurations and systematically compared the different methods. We then derived two non-linear configurations and a two-pass approach. We evaluate all pipeline configurations over four datasets of scholarly figures of different origin and characteristics. The quality of the extraction results is assessed using F-measure and Levenshtein distance, and we measure the runtime performance. Our experiments showed that there is a linear configuration that overall shows the best text extraction quality on all datasets. Further experiments showed that the best configuration can be improved by extending it to a two-pass approach. Regarding the runtime, we observed huge differences from very fast approaches to those running for several weeks. Our experiments found the best working configuration for text extraction from our method

Falk Böschen
Kiel University, Kiel, Germany
E-mail: fboe@informatik.uni-kiel.de
ORCiD: 0000-0003-4223-5353

Tilman Beck
Kiel University, Kiel, Germany
E-mail: stu127568@informatik.uni-kiel.de

Ansgar Scherp
ZBW - Leibniz Information Centre for Economics, Kiel, Germany
E-mail: a.scherp@zbw.eu
ORCiD: 0000-0002-2653-9245

set. However, they also showed that further improvements regarding region extraction and classification are needed.

Keywords Scholarly Figures · Text Extraction · Comparison · Figure Search

Acknowledgements This research was co-financed by the EU H2020 project MOVING (http://www.moving-project.eu/) under contract no 693092. We thank ABBYY Europe GmbH for providing us with a test license of the ABBYY FineReader for our experiments.

1 Introduction

Scholarly figures are data visualizations in scientific papers such as bar charts, line charts, and scatter plots [10]. Different research has been conducted to extract and use text from figures like translating the text to Braille [18], re-engineering the raw data from the figures [27], or for image search [29]. Many approaches follow a semi-supervised text extraction approach [9, 27]. However, semi-supervised approaches do not scale with the amount of scientific literature published today. Thus, unsupervised methods are needed to address the task of text extraction from scholarly figures. This task is challenging due to the heterogeneity in the appearances of the scholarly figures such as varying colors, font sizes, and text orientations. Nevertheless, extracting text from scholarly figures, such as the examples shown in Fig. 1, is an important task as the text provides additional information that is not contained in the papers [4]. To the best of our knowledge, we have recently performed the very first comparison of the different approaches for text extraction from scholarly figures [3]. The most likely reason for the lack of a comparative study is that existing works come from various different research areas. We extend our initial comparison by taking a commercial OCR engine into consideration as well as evaluating non-linear configurations and a two-pass approach. In addition, we describe the pipeline, the methods, and the configurations in more detail and perform a brief runtime analysis.

In total, we have investigated 7 different pipeline configurations motivated by approaches described in the literature. Each configuration is a combination of six to nine methods for the sequential steps in the extraction pipeline. Furthermore, we have created 37 modifications of the best performing pipeline configuration to systematically measure the influence of different methods applied in the text extraction pipeline. Thus, in summary, we have compared 44 different linear configurations for text extraction from scholarly figures in this paper. Additionally, we have evaluated two non-linear configurations as well as a two-pass approach. We assess each pipeline configuration with regard to the accuracy of the text location detection via precision, recall, and F1-measure. In addition, we evaluate the text recognition quality using Levenshtein distance. Finally, we are comparing the runtime of the different pipeline steps to find the most efficient configuration. We use four datasets in our evaluation which we made available recently[1]: one from economics [2] (EconBiz), one synthetically generated [19] (CHIME-S), one scanned and collected on the Internet [30] (CHIME-R), and one created from figures of academic books provided

[1] http://www.kd.informatik.uni-kiel.de/en/research/software/text-extraction, last access: September, 2017

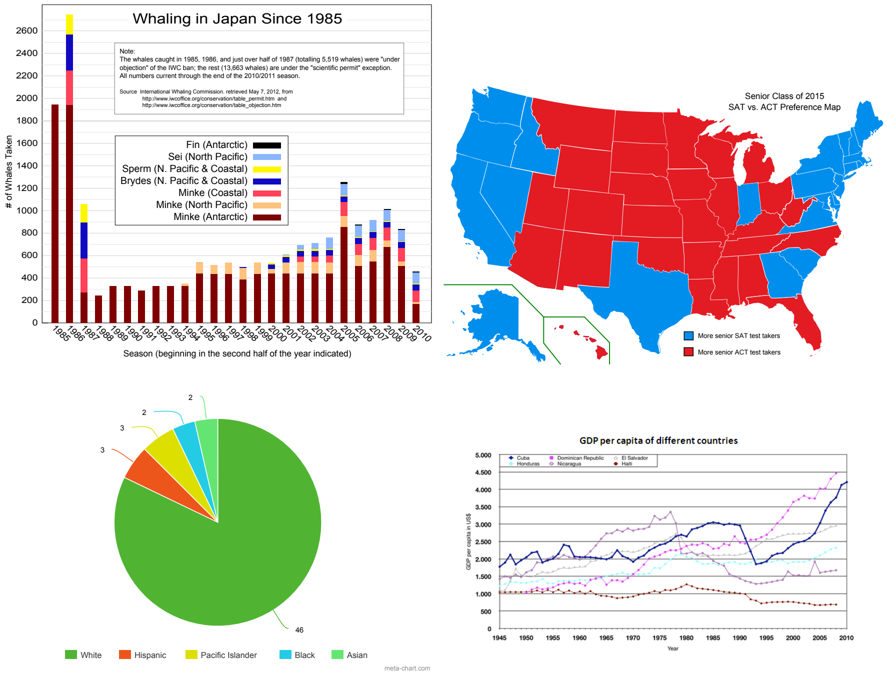

Fig. 1 Exemplary scholarly figures (Source: Wikimedia Commmons (Public Domain))

by the publisher DeGruyter[2] (DeGruyter). We manually labeled the EconBiz dataset and DeGruyter dataset, while the CHIME datasets were created in 2006 by the Center for Information Mining and Extraction, School of Computing, National University of Singapore. In summary, the contributions of the paper are:

(i) We conduct a systematic comparison of in total 44 linear configurations of a generic pipeline for text extraction from scholarly figures. Each configuration consists of a combination of six to nine methods from a total of 22 different methods that we have implemented.

(ii) We derive two non-linear configurations and a two-pass approach after analyzing the linear configurations and we compare them with the best linear configuration.

Please note that we also have made available[3] the implementation of our linear pipeline, including 21 methods, as well as 32 linear configurations for text extraction from scholarly figures that we compared. The subsequent section discusses the related work in the field. It serves as the foundation for defining our generic pipeline and the different methods used in the pipeline steps, as described in Section 3. In Section 4 we introduce the linear, non-linear, and two-pass configurations that we chose for evaluation. Section 5 describes the four datasets and the measures used in our

[2] `http://www.degruyter.com/`, last access: September, 2017

[3] `http://www.kd.informatik.uni-kiel.de/en/research/software/text-extraction`, last access: September, 2017

evaluation. The results are described in Section 6 and discussed in Section 7 before we conclude in Section 8.

2 Related Work

Text extraction from scholarly figures is addressed by research groups from different domains. Thus, one finds different terms that basically describe the same concept, such as information graphics [4], figures [10], charts [16], diagrams [6], and different variations of them. In the following, we will commonly denote them as scholarly figures or just figures. First, we discuss the relevant related work on text extraction from scholarly figures. Subsequently, we consider cartographic maps, domain-specific approaches from life sciences and chemistry, as well as briefly discuss approaches for making figures accessible to visually impaired users as well as applying text extraction on natural photos.

Scholarly Figures An early work on text extraction from scholarly figures is by Huang et al. [15, 16]. Their text extraction pipeline starts with a Connected Component Labeling (CCL) [25] that generates components of coherent text elements and graphics elements. In a subsequent step, these elements are separated by applying a series of filters. In the next step, the text elements are grouped using a derivation of Newton's formula for "Gravity" from classical physics. The authors claim that this method is capable of separating text at different orientations into different groups of text elements. Optical character recognition (OCR) is applied on these text groups and the recognized text is classified into strings and numbers. Finally, the recognized text is manually corrected in order to have a clean assignment to the corresponding graphical elements. Sas and Zolnierek [26] propose a three-stage approach for text extraction from figures. Starting with a conversion of the input image to gray-scale, the authors apply filtering operations, binarization, and CCL to generate coherent regions. Regions are filtered by empirical thresholds and classified into text and graphic elements using a decision tree. Tesseract[4] is used for OCR. Besides normal orientation, the input to the OCR engine is also rotated at a 90° angle to capture vertical text elements such as labels of the y-axis. Finally, the text detection is verified by assessing the number of special characters recognized in the text regions. Unfortunately, the authors did not assess the quality of their OCR results. Finally, we have developed a pipeline called TX for unsupervised text extraction from scholarly figures [1–3]. The TX pipeline uses an adaptive binarization method based on Otsu's method [24]. Subsequently, CCL is applied to extract coherent regions. A few heuristic rules are applied before the regions are clustered using DBSCAN in order to separate text elements from graphical elements. A Minimum Spanning Tree (MST) clustering is applied to detect single text lines. The orientation of these text lines is computed using a discrete Hough transformation and each line is rotated into horizontal mode in order to send it to a standard OCR engine. Here, the Tesseract OCR engine is used.

Cartographic Maps Cartographic maps use text elements to show city and street names, regions, and landmarks. An early work on text extraction from maps is the

[4] `https://github.com/tesseract-ocr/`, last access: September, 2017

approach by Deseilligny et al. [11] which relies on CCL for extracting regions. However, in contrast to the works on scholarly figures, Deseilligny et al. normalize each region in order to apply a rotation invariant character recognition. Multiple character hypotheses are generated for each region and those hypotheses are selected which create coherent strings and follow specific syntactic rules. A more recent approach is the semi-automatic text extraction proposed by Chiang et al. [9]. In contrast to most of the other works, the input image is not converted to gray-scale. Instead, a color quantization algorithm is applied. The authors separate text elements from graphical elements using a run-length smoothing algorithm based semi-automatic extraction that requires a positive and a negative example for each text-color/background-color combination. Text lines are detected by applying dilation operators on the connected components. The orientation of each line is estimated using a Single String Orientation Detection algorithm, which is based on morphological operations. The algorithm evaluates all possible orientations of text line candidates in a brute-force manner. The text line is rotated to horizontal orientation and ABBYY FineReader[5] is applied for OCR. After the OCR phase, a recognition confidence score is computed to filter the results.

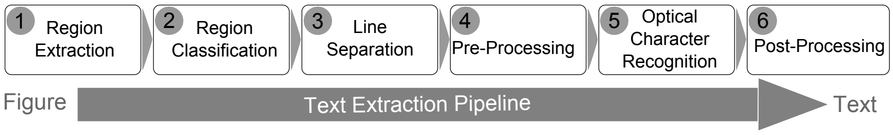

Fig. 2 Generic linear pipeline for text extraction from scholarly figures abstracted from the literature

Domain Specific Text Extraction An algorithm for text detection in biomedical images, as part of the Yale Image Finder, was proposed by Xu and Krauthammer [29]. The authors first detect and remove so-called layout elements, followed by a binarization, median filter, and edge detection with the Sobel operator. The text region extraction, based on horizontal and vertical histogram projection analysis, is conducted on the edge image. This is performed recursively until the image cannot be split any further. During this recursive processing of the regions, heuristic filters are applied to only subdivide those regions that contain text and discard the others. Lu et al. [22] developed a retrieval engine for scholarly figures in chemistry. First, the input image is converted to gray-scale and an edge image is computed. A Hough transformation is applied to the edge image in order to compute a feature vector. The feature vector is used to classify the input in order to find 2D plots. Only 2D plots are further processed. First, the 2D plot is binarized. Then, the axes are detected and the plot is segmented by applying CCL. The text detection is based on fuzzy rules and includes a method for separating overlapping characters. The recognition of text strings is conducted using GOCR[6].

[5] `http://www.abbyy.com/ocr-sdk/`, last access: September, 2017

[6] `http://www-e.uni-magdeburg.de/jschulen/ocr/index.html`, last access: September, 2017

Access for the Visually Impaired Another approach that requires text extraction from figures is the work by Jayant et al. [18]. Their goal is to translate figures into Braille language for the visually impaired. First, a color reduction is conducted with Adobe Photoshop. Subsequently, the figure is manually classified into a set of predefined figure types. CCL is applied to the figure to extract regions. In order to separate text elements from graphical elements, the authors manually train a Support Vector Machine (SVM) per figure type as well as per book where the figures were taken from. Thus, the authors make the assumptions that all figures of a certain type have a similar design throughout a single book. Subsequently, a separation into text line structures is performed, using a so-called label training algorithm which uses a Minimum Spanning Tree with manually created test data. The text line orientation is estimated by minimizing the perpendicular squared distance. Finally, OCR is conducted with Omnipage[7] or ABBYY FineReader. Carberry et al. [5] analyze figures, especially bar charts, pie charts, and line charts, to generate textual summaries for visually impaired users. Their Visual Extraction Module (VEM) claims to be capable of extracting text elements and their position [7]. However, the paper does not provide technical details on how this is achieved. In their current work [5], manually generated datasets are used instead of the VEM, which may indicate that the VEM does not generate an output of sufficiently high quality.

Approaches on Natural Photos Besides text extraction from scholarly figures and related images, there is also research regarding OCR on natural photos. The research in this area has several interesting ideas to solve the OCR problem. For example, Olszewska [23] presented a template-matching approach to extract numbers of arbitrary position and orientation in the captured 3D space in images using contour information. Other approaches show promising results as well [12,21]. However, the works for text extraction from natural photos often make specific assumptions about the difference in the appearance of text and background/graphic elements. For example, the assumptions that text elements are generally smaller than graphic elements [14], that text elements can be identified via their edges [21], or that text has a unique or more homogeneous color [14]. These assumptions often do not hold for scholarly figures like charts, diagrams, or graphics.

3 Generic Pipeline for Text Extraction

Based on the related work, we derived a generic pipeline for text extraction from scholarly figures. The pipeline consists of six steps as illustrated in Fig. 2. Each step can be implemented by different methods. A combination of methods along the pipeline steps is called a configuration. We can perform a fair comparison of different approaches and methods by comparing different configurations of the pipeline. Below, we provide a brief summary of the different steps that we identified for the generic pipeline. A formalization of the pipeline can be found in our earlier publication [1]. This pipeline is the basis for our linear configurations as well as, in a mod-

[7] `https://www.nuance.com/print-capture-and-pdf-solutions/optical-character-recognition/omnipage/omnipage-server-for-developers.html`, last access: September, 2017

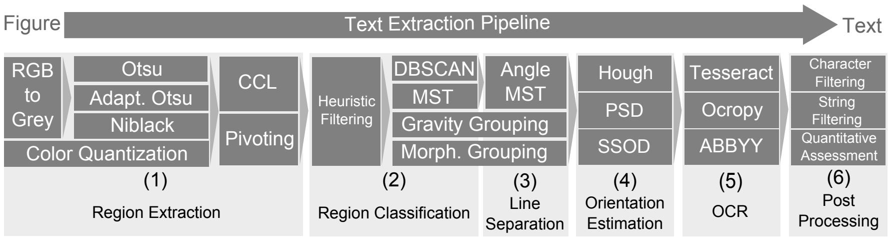

Fig. 3 Overview of the possible linear pipeline configurations showing the different methods for each pipeline step

ified form, for our non-linear and two-pass configurations. In Section 4, we discuss the different configurations of the pipeline assembled from the methods described below. An overview of the possible linear configurations can be found in Fig. 3.

3.1 Definition of the Generic Pipeline

Please note, for describing the steps of our generic pipeline we use the following terminology: We refer to scholarly figures as **images** since it is the accepted term in computer vision. A **region** is a set of pixels of an image. Each region constitutes either one or sometimes multiple text characters or graphical symbols. A **text line** or text element is a set of regions representing text.

The input to the pipeline is a (color) image of a scholarly figure and the output are text elements together with their position, dimension, and orientation. The six steps are as follows: **(1)** The first step extracts regions from an image. Thus, a decision on pixel-level has to be made about what part of the image belongs to a region and what is background. A common algorithm for this task is Connected Component Labeling (CCL) on a binarized image. **(2)** In the second step, the previously computed regions are classified either as text or graphics. The regions classified as graphics are ignored in the subsequent steps. **(3)** The third step computes text lines from the text regions provided by the previous step. It is necessary to compute text lines since most OCR engines work only on horizontal text input and the orientation can be estimated best from single text lines. **(4)** The fourth step estimates the orientation of the text lines and performs other pre-processing that a specific OCR engine might need. Besides rotating text lines to horizontal orientation, one may need to scale text lines to a sufficiently high resolution or remove noise. **(5)** Subsequently, the fifth step actually performs the Optical Character Recognition (OCR). A commonly used OCR engine is Tesseract, developed by Google and used in the Google Books project. **(6)** Finally, post-processing is applied to the OCR results. For example, the OCR output is corrected using some heuristics.

3.2 Methods of the Six Pipeline Steps

For each step of the generic pipeline, we compare different methods motivated by approaches described in the literature. Below, we describe the methods selected along the steps of the generic pipeline as shown in Fig. 2. The large number of methods allows only a brief description of each method, but further details can be found in the references.

3.2.1 Step (1): Region Extraction

The region extraction step consists of two sub-steps: First, the input is transformed from color space into one or multiple binary images as described below. Second, the actual region extraction is conducted using one of two approaches that we identified in the literature. Overall, the output of this step is a set of binary regions where each region represents one or more text characters or graphical symbols.

Binarization of Color Images Given a color image, one can convert it directly to multiple binary images. An alternative is to use an intermediate transformation to a grey-scale image, which is then converted to the binary output. For directly converting a color image to multiple binary images, a so-called **Color Quantization** method can be used. Color Quantization performs a clustering over the color space. For each cluster, a representative color is chosen and each color in the original image is replaced by the color of the cluster closest to it. Finally, the image is split into multiple binary images, one for each color, where all pixels that have the specific color are set and the others are not. The color quantization method is inspired by the work of Chiang [8], Fraz [12], and Jayant [18]. The **RGB to Grey via Luminance** method is commonly used to create an intermediate grey-scale image [2, 15]. It uses the conversion formula $Y = 0.2126R + 0.7152G + 0.0722B$ to weight the color components red (R), green (G), and blue (B) to the luminance value Y based on human perception. Subsequently, the grey-scale image is binarized using one of the following methods: **Otsu's Method** [24] separates two regions by finding the threshold that maximizes the inter-class variance. One problem with Otsu's method is that it only computes one threshold (or multiple if the Multi-Otsu method is used) which are applied globally on the entire image. This leads to problems with local inhomogeneities like varying text-color/background-color combinations. In order to address this challenge, an **Adaptive Otsu Binarization** method [2] was developed that computes multiple thresholds to create a locally adaptive binarization. This is achieved by subdividing the image into smaller parts and recursively applying Otsu's method to compute new thresholds. **Niblack's Method** and its variants are other options for binarization [20]. We tested all versions of Niblack's Method as described by Khurshid et al. [20] with the specified parameters. The modified version of Sauvola performed best during our preliminary tests. Thus, we compare it with the two Otsu variants.

Region Extraction from Binary Images Given a binary image, we can apply one of the following two methods for region extraction: The most common method [2, 15, 26] is **Connected Component Labeling (CCL)** [25]. The CCL algorithm iterates over the whole image and assigns each pixel to a region by taking the assignment of the adjacent pixels into account (a 4- or 8-pixel-neighborhood). As an alternative, we use Xu and Krauthammer's [29] **Pivoting Histogram Projection** method for region extraction. Here, first, an edge image of the binary image is computed. The edge image's pixels are alternately projected on the x- and y-axes and the image is split after every projection at the minimal point(s) of the histogram into multiple sub-images. Each sub-image is further processed while alternating the direction until no further split is possible. Thus, one obtains multiple rectangular areas, which are converted into regions by taking all foreground pixels of each region from the binary image.

3.2.2 Step (2): Region Classification

The output of the region extraction in Section 3.2.1 has to be classified into text elements and graphical elements. For each region, we compute a feature vector which is composed of the center of mass x-/y-coordinates, width and height, and area-occupation-ratio (the number of foreground pixels of the bounding box divided by the area) that are used to group the regions into text and graphics. **Heuristic Filtering** methods are commonly applied as pre-processing to help separate text elements from graphical elements [2,26]. For example, very small regions typically constitute noise, large regions are graphical elements like axes, and average-sized regions refer to characters. Another approach is to consider the coverage of the bounding box of a region. Heuristic filters are parameterized and thus require a suitable choice of parameter values. For region classification, unsupervised density-based clustering algorithms like **DBSCAN** can be used to group regions [2]. Since text is normally denser and of a different size than graphic elements, it can be separated from graphic elements using DBSCAN. Another method to distinguish between text and graphics is the graph-based **Minimum Spanning Tree (MST)** clustering algorithm [18]. After constructing the tree, the clusters are created by splitting the graph, i. e., removing edges. There are multiple possibilities on how to split the tree. In our experiments, we consider splits at inconsistent edges, i. e., edges that are longer than the local average edge length. Huang et al. [16] proposed **Grouping Rules based on Newtons Gravity Formula** from classical physics. The formula computes a threshold which defines whether two regions are grouped together. This results in text elements, each representing a single text line. Finally, the **Morphological Method** by Chiang et al. [4,9] is applied to the individual pixels of the image and uses morphological operations to merge characters into words. Thus, it also generates text elements.

3.2.3 Step (3): Separation into Text Lines

The methods for determining text elements in the previous step can result in clusters of regions of various shape. Thus, the text elements may contain text of different orientation. This is problematic for standard OCR engines as they require text at a horizontal orientation. Therefore, it is necessary to split the text elements into single text lines, since one can only reliably estimate the orientation for single lines of text. One method to separate text elements into text lines is to apply an **Angle-Based MST** [2] clustering. The MST is constructed over the centers of mass of the regions in a cluster. The assumption is that characters of a text line are closer to each other than characters from different text lines. Thus, most edges of the MST constitute a single line while only a few edges connect across different text lines. Edges connecting different lines will have orientations that differ strongly from the main orientation and can therefore be easily removed.

3.2.4 Step (4): Text Line Orientation

As said above, standard OCR engines require that the text of the input image has a horizontal orientation. In this step, the single text lines produced in the previous step are analyzed with regard to their orientation. We compute the orientation of each line using one of the following methods: The first method uses the **Hough Transformation** [17] to calculate the orientation [2]. The method transforms all center of mass coordinates of the characters of a text line into Hough space and the maximal value in this Hough space represents the main orientation. Since text lines can have only an orientation between -90 and +90 degree, we can limit the Hough space computation to this interval. Another method to estimate the orientation is to minimize the **Perpendicular Squared Distance** of the bounding box of each text line [18]. The **Single String Orientation Detection (SSOD)** [9] method assesses different orientation candidates by rotating the text elements and applying morphological operations on its regions. The orientation at which the largest pixel area remains after applying the operators is selected as the orientation of the text element. Subsequently, we rotate the text lines into the opposite direction to bring them into horizontal alignment.

3.2.5 Step (5): Optical Character Recognition

We now have individual text lines at a horizontal orientation. Thus, in this step, we can apply standard Optical Character Recognition (OCR) engines to extract the text. We analyzed different OCR engines mentioned in the related work. We have selected Tesseract and Ocropy[8] as they are freely available and frequently updated. We also select the commercial OCR engine ABBYY FineReader for comparison. The OCR engine **Tesseract** provides trained models for different languages, where we choose English. We are not using Tesseracts Layout Analysis capabilities for removing graphic elements and conducting line detection since it only works on horizontal text. Tesseract has been used in the past for text extraction from scholarly figures [2, 26]. **Ocropy** is a collection of open source tools for document analysis and OCR. It is designed for character recognition from full-page documents like Tesseract. Ocropy has several constraints regarding parameters like minimum image width and height in order to assure good results. Thus, we modify the input image to fulfill the required thresholds by properly scaling the text lines. Like Tesseract, the OCR engine Ocropy provides a pre-trained model for the English language. The commercial OCR engine **ABBYY FineReader** is widely used and offers SDKs for various platforms besides ready to use desktop applications. Similar to Tesseract, ABBYY FineReader is able to recognize several languages, but we limited the recognition language to English. A license is required to use the ABBYY FineReader and ABBYY provided us with a developer license for the ABBYY FineReader 11 for our experiments. Similar to Tesseract, ABBYY FineReader can only recognize horizontal text.

[8] `https://github.com/tmbdev/ocropy`, last access: September, 2017

3.2.6 Step (6): Post-Processing

The last step of the pipeline conducts potential corrections of the textual output of OCR engines. Here, several approaches exist: For example, one can use dictionaries [28] to correct the OCR output. Since text in scholarly figures is very sparse and often contains abbreviations and numbers, one cannot apply standard dictionaries. A much simpler heuristic-based correction is the **Special Character Filtering per single Character** method. It removes all special characters from the output, i. e., all characters that are not a white space, number, or character from a-z or A-Z. This makes sense because recognition errors often appear in form of special characters like dots or dashes in the output. Sas and Zolnierek proposed a **Special Character Filtering per String** [26], which is a modified version of the previous method. Here, complete text elements are removed if they contain too many special characters. Another post-processing method is the **Quantitative OCR Assessment** by Chiang et al. [9]. The main idea is to reuse knowledge from previous steps of the extraction pipeline by comparing the number of regions that went into the OCR process with the number of characters that were recognized from them. If the difference is above a certain threshold, one can assume that one or multiple recognition errors happened during the OCR process. While Chiang's approach also takes recognition confidence information on character level from ABBYY FineReader into account, one cannot, in general, assume that this information is available from all OCR engines. Thus, the implemented method performs its post-processing only based on the difference between the number of regions before and the number of characters after the OCR step.

4 Pipeline Configurations

From the methods defined in the previous section, one can create various pipeline configurations. Some methods are restricted in how they can be combined as illustrated in Fig. 3. Section 4.1 discusses the linear configurations that are motivated from the literature, followed by a description of the systematically modified linear configurations. Section 4.2 presents the non-linear configurations and Section 4.3 describes the two-pass configurations.

4.1 Linear Pipeline Configurations

In the following, we describe the configurations motivated from the literature as well as the systematic modifications to them.

Configurations motivated from the Literature

There are seven pipeline configurations that are motivated from the literature. Each configuration is identified by *(x)*, an acronym created from the contributing author(s) and the publication year. The first configuration *(SZ13)* is inspired by the work of Sas and Zolnierek [26]. It uses Otsu's method for binarization, followed by CCL. Subsequently, it applies heuristic filtering similar to the original approach. The decision tree used by Sas and Zolnierek is replaced by the line generation approach based on MST. The rationale behind this is that a decision tree is a supervised method while MST is unsupervised. This configuration does not use any method for orientation estimation from step 4 of the pipeline since the original work by Sas and Zolnierek does not have such a feature. Tesseract is used as OCR engine since it was also used in the original paper. In the post-processing step, all strings that contain too many special characters are removed. The second configuration *(Hu05)* is based on the work of Huang et al. [16]. After region extraction using Otsu binarization and CCL, the Heuristic Filter method is applied, and the regions are grouped using the Gravity method. Finally, the grouped regions are processed with Tesseract. Based on the work of Jayant et al. [18], configuration *(Ja07)* starts with Otsu's method and CCL. Subsequently, it clusters the regions using an MST and approximates the orientation by minimizing the perpendicular squared distance. Text recognition is achieved by applying Tesseract.

Different from the previous configurations, the fourth configuration *(CK15)* – inspired by Chiang et al. [9] – uses Color Quantization to generate multiple binary images, followed by a CCL. Subsequently, it applies heuristic filtering and Morphological Clustering on the regions. This step differs from the original paper, where the relevant color levels were manually selected. Thus, we assess all extracted binary images. The orientation of each cluster is estimated using the SSOD method, followed by Tesseract OCR, and quantitative post-processing. Similar to the previous pipeline configuration, the fifth configuration *(Fr15)*, inspired by Fraz et al. [12] from the photo processing domain, starts with Color Quantization and CCL. The original approach uses a supervised SVM to form words, which we replaced with unsupervised methods from our methods set. The extracted regions are filtered and DBSCAN is applied, followed by an MST clustering into text lines. The orientation of the text lines is calculated using Hough method and the text is recognized using Tesseract. All configurations so far use CCL to extract regions.

The sixth configuration *(XK10)*, motivated by Xu and Krauthammer [29], uses the pivoting algorithm after binarization with adaptive Otsu. The regions are filtered using heuristics and grouped into lines using DBSCAN and MST. This differs from the original work, which only applied heuristic filtering to remove the graphic regions. The reason behind this is that the authors only aimed at finding text regions and not to recognize the text. Thus, we filled the rest of the pipeline steps with suitable methods. The orientation of each line is estimated via Hough and OCR is conducted with Tesseract.

Finally, configuration *(BS15)* resembles our own work [2]. It uses adaptive Otsu for binarization and CCL for region extraction. Heuristic Filtering is applied to the regions and DBSCAN groups them into text elements. Text lines are generated using the angle-based MST approach and the orientation of each line is estimated via Hough transformation, before applying Tesseract's OCR.

Configurations resulting from Systematic Modifications

In order to evaluate the influence of the individual methods, we chose the pipeline configuration *(BS15)* as the basis for our systematical modifications, since it is the most recent development for the task of automatically extracting text from scholarly figures and showed the best performance of the seven configurations from the literature. The systematic modifications are organized along the six steps of the generic pipeline in Figure 2. Each of the systematic configurations has an identifier *(BS-XYZ)* based on the original configuration, where X is a number that refers to the associated pipeline step and YZ uniquely identifies the method. We systematically modified the configurations in the following way:

In the first step, the binarization and region extraction is evaluated with the following configurations: *(BS-1NC)* differs from *(BS15)* by using Niblack instead of adaptive Otsu for binarization. Configuration *(BS-1OC)* uses the third option for binarization, Otsu's method. Color quantization is combined with the pivoting region extraction in *(BS-1QP)*. The second and third step for region classification and generation of text lines is assessed with the following configurations: Configuration *(BS-2nF)* differs from the base configuration by not applying the optional heuristic filtering method. Configuration *(BS-2CG)* uses the Gravity Grouping instead of DBSCAN and MST. Configuration *(BS-2CM)* applies MST to cluster regions and create text lines. Morphological text line generation is used in configuration *(BS-23M)*. The following two configurations are used to evaluate the other options for estimating the orientation of a text line: Configuration *(BS-4OP)* uses the Perpendicular Squared Distance method and configuration *(BS-4OS)* uses the Single String Orientation Detection method to estimate the orientation. In all configurations, both open source OCR engines are used to generate the results. We add an identifier to a configuration when referencing a configuration with a specific OCR engine. Furthermore, we assess the direct impact of the OCR engine on the recognition results with configuration *(BS15-O)*, which only differs with respect to the OCR method from the base configuration by using the Ocropy OCR engine instead of Tesseract. Similarly, we have evaluated most of the systematic configurations as well as the base configuration with the ABBYY FineReader OCR engine. The sixth and last step of the pipeline is the post-processing. We use three configurations to evaluate the different post-processing methods: First, configuration *(BS-6PC)* uses the Special Character Filter method for post-processing. The second configuration *(BS-6PS)* uses the String Filter method for post-processing. The third configuration *(BS-6PQ)* uses the Quantitative Assessment method for post-processing.

4.2 Non-Linear Pipeline Configurations

Our text-extraction framework is designed as a linear processing pipeline as described in Section 3.1. However, researchers have also successfully applied non-linear systems when processing images [13]. The idea is to improve the performance by reusing information about the structure and content of the image, which is gained in early steps of the pipeline and use this information in later steps or by repeating earlier steps. This is not possible with the linear configurations discussed so far. Thus, we analyzed the pipeline to identify the options for non-linear configurations. A very important information is the location of text inside a figure. Therefore, one may refine the results from certain parts of a figure if the quality of the recognition in the OCR step makes it necessary. Our non-linear configurations are based on the previous best configuration *(BS-4OS)* which is also used for comparison. The non-linear configurations check the recognized text after step 5 of the pipeline for special characters. If such a character is present in the recognized text string, then the pipeline is repeated on the processed sub-image of the initial input image. We consider two different kinds of pipelines for repetition which are described next.

An important step for text extraction is the correct binarization of the image, which is part of the first step of our pipeline. Our first non-linear configuration *(BS-4OS/R1)* aims at improving the results by applying the binarization a second time onto the already extracted text regions. Thus, it repeats the binarization step before executing the OCR step a second time. Here, we use standard Otsu binarization and not the adaptive Otsu, since we have only small parts of the original image.

The second non-linear configuration *(BS-4OS/R2)* is also based on configuration *(BS-4OS)*. The configuration repeats the whole pipeline on the extracted text regions which contain special characters. Thus, we assess the text output after the OCR step and apply the region extraction, region classification, separation into text lines, and text line orientation estimation again.

4.3 Two-Pass Pipeline Configurations using ABBYY FineReader OCR

Correctly extracting rotated text from figures is one of the main challenges for a good extraction result. We conducted several experiments using the best linear configuration *(BS-4OS(Ocropy))* and the standalone ABBYY FineReader separately to compare the OCR output of figures containing only horizontal text and figures with only rotated text. Based on the DeGruyter and EconBiz datasets (see Section 5.1) we created two different datasets with only horizontal and only rotated text using the orientation information of the gold standard. For the dataset containing only figures with rotated text, we removed every text area with an angle between -2 and 2 degrees. Accordingly, for the other dataset, all text areas that do not have an orientation in the range of -2 to 2 degrees were removed.

The results showed that the ABBYY FineReader OCR engine can extract horizontal text very well. However, graphic elements and rotated text are problematic and are falsely recognized. For the rotated text, the linear configuration *(BS-4OS(Ocropy))* achieves better results. Detailed results of the experiments are shown in Table 1. Please refer to Section 5.2 for an explanation of the measures used for comparison.

These results suggest a two-pass processing pipeline that first applies ABBYY FineReader to extract the horizontal text and then uses the pipeline to extract the rotated text. After extracting the horizontal text, the figure is manipulated by filling the corresponding bounding box of the text area with the color white. Then, the best linear configuration *(BS-4OS(Ocropy))* is applied on the manipulated figure to extract the remaining text. However, a decision has to be made about which output of

Table 1 F1-measure ($F1_D$) for Text Location Detection, F1-measure ($F1_C$) for Text Element Coverage, Average Local Levenshtein (AVG_L), Average Global Levenshtein (AVG_G), and operations per character (OPC) for standalone ABBYY FineReader and the configuration *(BS-4OS)* using Tesseract as well as Ocropy as OCR engine.

Config.	$F1_D(SD)$	$F1_C(SD)$	$AVG_L(SD)$	AVG_G	OPC
only horizontal text					
ABBYY FineReader	0.75 (0.20)	0.63 (0.16)	2.02 (2.61)	44.86	0.28
BS-4OS (Tesseract)	0.70 (0.20)	0.64 (0.12)	4.03 (3.21)	70.26	0.51
BS-4OS (Ocropy)	0.68 (0.21)	0.55 (0.18)	3.28 (3.18)	69.57	0.49
only rotated text					
ABBYY FineReader	0.27 (0.21)	0.33 (0.15)	12.59 (9.43)	84.97	1.58
BS-4OS (Tesseract)	0.49 (0.29)	0.64 (0.21)	17.46 (14.74)	82.01	1.76
BS-4OS (Ocropy)	0.47 (0.28)	0.40 (0.22)	11.75 (9.84)	72.51	1.03

ABBYY FineReader in the first step is acceptable since it also falsely recognizes non-horizontal text and graphic elements. We use the confidence value per word which is provided by the ABBYY FineReader SDK[9] to decide which text line to accept from ABBYY FineReader. According to the provided documentation, it is calculated using different bonuses (e. g. recognized word from a dictionary) and penalties (e. g. bad recognition quality). We don't provide any supplementary dictionary or word model to the engine. Furthermore, we use the more accurate calculation of confidence offered by ABBYY FineReader, which comes at the cost of a slower recognition speed.

If the confidence value exceeds a predefined threshold, we accept the recognized output and remove the text area from the figure. Recognized lines of text often contain multiple words. Thus, we offer a configuration parameter deciding whether we take the average confidence value of all words of the line or the minimum confidence value of all words of the line. The latter produces higher quality output while accepting fewer text lines from the output due to poorly recognized single words within a text line. We evaluate two configurations *(BS-4OS-TP-0avg)* and *(BS-4OS-TP-0min)* to test both options.

For standard document pages, the confidence values are in the range from 0 to about 100 as can be seen in Fig. 4. In contrast, the confidence values of the text elements recognized in figures can also be negative. Investigations of the produced confidence values suggested using a threshold of zero for both text line confidence values since we observed that short words and numbers were assigned low confidence values close to zero even though they were correctly recognized. Non-horizontal text or certain graphic elements (e. g. dashed lines) have larger negative confidence values whereas correctly recognized text elements mostly result in high positive values.

5 Evaluation

We conduct our evaluation on four datasets which are described in Section 5.1. The evaluation measures are described in Section 5.2.

[9] `https://www.abbyy.com/en-us/ocr-sdk/`, last access: September, 2017

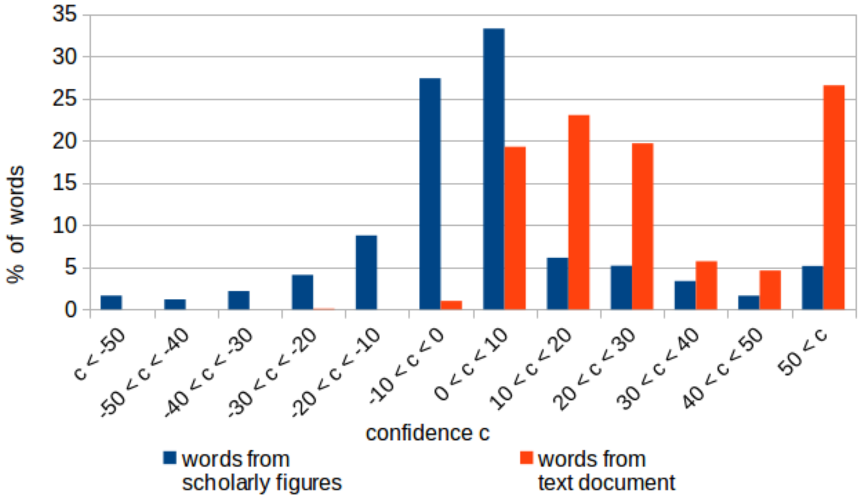

Fig. 4 Comparison of the confidence values of words from scholarly figures and words from a text document

Table 2 Number of figures, average figure width and height, and average number of text elements (TE), words, and characters per figure.

Dataset	# Figures	Width	Height	# TE	# Words	# Characters
EconBiz	121	982	681	25	35	151
DeGruyter	120	959	619	24	34	149
CHIME-R	115	714	454	14	18	69
CHIME-S	85	440	320	12	18	76
Total	441	801	535	19	27	114

5.1 Datasets

We have created two datasets using our own tool that was specifically designed for manually labeling text elements in figures. One dataset is in the domain of economics, the other is created from educational books. In addition, we use the CHIME datasets, which consist of figures of varying quality and origin.

- **EconBiz** We have created a corpus of 121 scholarly figures from the economics domain. We obtained these figures from a corpus of 288,000 open access publications from EconBiz[10] by extracting all images, filtering them by size, ratio, dominant color, and others. We randomly selected a subset of 121 figures, which resemble a wide variety of scholarly figures from bar charts to maps. The figures were manually labeled to create the necessary gold standard information.

[10] `https://www.econbiz.de/`, last access: September, 2017

- **DeGruyter** We manually labeled another dataset, composed of scholarly figures from books provided by DeGruyter[11] under a creative commons license[12]. We selected ten books, which contain figures with English text and selected 120 figures randomly from these books. Most of the selected books are from the chemistry domain. The gold standard for these figures was created using the same tool which has been used for the creation of the EconBiz dataset.
- **CHIME-R** The Chart Image Dataset[13] consists of two subsets. The CHIME-R consists of 115 real images that were collected on the Internet or scanned from paper. Most of the figures are bar charts. The other figures are pie charts or line charts. The gold standard was created by Yang Li [30].
- **CHIME-S** The other, CHIME-S dataset consists of 85 synthetically generated images. This set mainly contains line charts and pie charts and few bar charts. The gold standard was created by Zhao Jiuzhou [19].

Some statistics about the datasets which are useful for understanding the evaluation results can be found in Table 2. Both, the CHIME-R and CHIME-S datasets contain figures with an on average lower resolution than the EconBiz and DeGruyter datasets, which are almost equal. With respect to the average number of characters, words, and text elements in a figure, the distribution of EconBiz and DeGruyter are similar, with about twice as many as the CHIME datasets.

5.2 Measures

We have selected four measures to evaluate the pipeline configurations and compare their results. Our gold standard consists of text elements which represent single lines of text taken from a scholarly figure. Each text line consists of one or multiple words which are separated by blank space. Each word may consist of any combination of characters and numbers. Every text line is defined by a specific position, size, and orientation.

Each pipeline configuration generates a set of text line elements as well. These text lines need to be matched to the gold standard. Since we do not have pixel information per character, we match the extraction results with the gold standard by comparing their bounding boxes. Thus, we determine the intersection of the bounding box provided by the gold standard with the bounding box extracted by our pipeline. If the overlap between the bounding boxes is larger than a certain percentage with respect to the overall area covered by the boxes, we consider the result as a match. This reduces the error introduced through elements which are an incorrect match and only have a small overlap with the gold standard. We empirically evaluated different intersection thresholds, ranging from 5% up to 30%. A threshold of 10% has shown good results regarding finding correct matches (true positive) and avoiding incorrect matches (false positive).

[11] `http://www.degruyter.com/`, last access: September, 2017

[12] `http://www.degruyter.com/dg/page/open-access-policy`, last access: September, 2017

[13] `https://www.comp.nus.edu.sg/~tancl/ChartImageDataset.htm`, last access: September, 2017

We look at each gold standard element and take all elements from the pipeline as matches that are above the intersection threshold. Thus, a gold standard element can have multiple matching elements and an element from the pipeline can be assigned to multiple elements from the gold standard if it fulfills the matching constraint for each match. We have defined three measures to assess these matches. The first two measures analyze the text localization. The third measure compares the recognized text. Finally, we also decided to analyze the performance of the individual methods.

Text Location Detection First, we evaluate how accurate the configurations are at detecting the text locations. If at least one match is found for an element from the gold standard set, it counts as a true positive, regardless of what text was recognized. If no match was found, it is considered as false negative. A false positive is an element from the pipeline output which has no match. From these values, we compute precision, recall, and F1-measure for assessing the text location detection. This measure is a binary evaluation and assesses only whether a match to an element exists or not. In addition, we report the Element Ratio (ER) which is the number of elements recognized by the pipeline divided by the number of elements in the gold standard and the Matched Element Ratio (MER) which is the number of matched items from the pipeline divided by the number of elements of the gold standard. These ratios give an idea whether gold standard elements get matched by multiple elements and whether the configuration tends to find more elements or less elements than it actually should find. We restrict the number of matches by applying a coverage threshold, which means that the intersection area of two elements has to be at least as big as 10% of the total covered area. This measure penalizes mappings between elements of largely varying size. We further evaluate these matches with the next two measures.

Text Element Coverage Second, we investigate the matching in more detail by assessing the text element coverage. For each gold standard text element, we take the pixels of the bounding boxes and compute their overlap to calculate precision, recall, and F1-measure over all of its matches. The true positives, in this case, are the overlapping pixels and the false positives are those pixels from the text elements from the pipeline which are not overlapping. The false negatives are the pixels of the gold standard element which were not covered by a text element from the pipeline. The values are averaged over all gold standard text elements in a figure.

Text Recognition Quality Third, we assess the quality of the recognized text by computing the Levenshtein distance between the extracted text and the gold standard. We calculate the distance for each match and report the average for the whole figure. Since multiple text elements from the pipeline can be matched to a gold standard text line, we have to combine their text into one string. We combine the elements using their position information. Besides a (local) Levenshtein Distance per match, we also compute a global Levenshtein distance over all extracted text. This means that for each figure, we combine all characters from the text elements of the gold standard and add them to one string. Likewise, we create a string from the text elements extracted by the pipeline. The characters in both strings are sorted alphabetically and we compute the Levenshtein Distance between these strings. This approximates the overall number of operations needed to match the strings without considering position information. Since the global Levenshtein Distance depends on the number of characters inside a figure, we also report an operations per character (OPC) score, which is computed by dividing the global Levenshtein Distance by the number of characters in the gold standard. This normalizes the global Levenshtein Distance and makes it comparable across scholarly figures with different amounts of characters.

Runtime Performance Finally, we log the execution time for each method in milliseconds to see which method has the best runtime performance. We analyze each configuration individually and compute the average time for each method over all figures from all four datasets. For each configuration, we aggregate the execution time of its methods to compute the average time needed per step of the pipeline.

6 Results

First, we present the evaluation results of the linear configurations in Section 6.1. In Section 6.2 we show the results for the non-linear configurations and finally, in Section 6.3, the results for the two-pass approach.

6.1 Linear Pipeline Configurations

We have executed all configurations listed in Section 4.1 and analyzed the output with respect to the measures for Text Location Detection, Text Element Coverage, Text Recognition Quality, and Runtime Performance as defined in Section 5.2. For reasons of simplicity, we are only reporting the average values over all datasets for all configurations as well as for each dataset separately. We compute the average precision, recall, and F1-measure over the elements of each figure. We report the average precision, recall, and F1-measure in terms of mean and standard deviation over all individual results per figure. The local Levenshtein distance is reported as the average of the mean values per figure and the average standard deviation. The global Levenshtein distance is defined by the mean and standard deviation over all figures and the normalized OPC score. Finally, we report the performance of the methods and configurations, measured over the execution time averaged over the images of all datasets.

First, we report the results for the measures Text Location Detection, Text Element Coverage, Text Recognition Quality, and Runtime Performance of the configurations from the literature. Subsequently, we present the results for the systematically modified configurations.

The text location detection results for the configurations from the literature computed over all datasets are reported in Table 3. The best F1-measure is achieved by configuration *(BS15)* with a value of 0.58. Looking at each dataset separately as documented in Table 4, one can see that configuration *(BS15)* works best for the De-Gruyter (0.70) and CHIME-R (0.63) datasets while *(SZ13)* has the best result for the CHIME-S (0.55) and EconBiz (0.57) datasets. The coverage assessment in Table 5 shows the best precision of 0.79 for *(Hu05)*, the best recall of 0.59 for *(SZ13)*, and the best F1-measure of 0.57 for *(Hu05)*. The individual results per dataset for all configurations are shown in Table 6.

The text recognition quality is presented in Table 7 and the individual results per dataset for all configurations are presented in Table 8. We obtain the best results with

Table 3 Precision (Pr), Recall (Re), and F1-measure for the Average Text Location Detection, Element Ratio (ER), and Matched Element Ratio (MER). Results are averaged over all datasets for configurations from the literature.

Config.	*Pr*	*Re*	*F*1 (*SD*)	*ER*	*MER*
SZ13	0.63	0.47	0.54 (0.23)	0.80	0.59
Hu05	0.61	0.43	0.48 (0.28)	0.77	0.57
Ja07	0.59	0.45	0.49 (0.28)	0.83	0.51
BS15	0.66	**0.55**	**0.58 (0.25)**	**1.04**	0.69
CK15	0.52	0.50	0.53 (0.23)	1.37	0.60
Fr15	0.55	0.51	0.54 (0.25)	1.44	**0.72**
XK10	**0.73**	0.35	0.45 (0.26)	0.43	0.39

(BS15) with 0.67 operations per character (OPC), an average local Levenshtein of 6.23, and an average global Levenshtein of 108.81. Only configuration *(CK15)* has a slightly better average local Levenshtein (6.07).

Table 4 Average F1-measure and Standard Deviation for Text Location Detection per configuration for the individual datasets EconBiz, DeGruyter, CHIME-R, and CHIME-S.

Config.	EconBiz	DeGruyter	CHIME-R	CHIME-S
SZ13	0.57(0.25)	0.51(0.24)	0.53(0.21)	0.55(0.23)
Hu05	0.45(0.29)	0.55(0.30)	0.50(0.25)	0.34(0.26)
Ja07	0.47(0.29)	0.50(0.27)	0.52(0.26)	0.33(0.22)
CK15	0.53(0.22)	0.54(0.21)	0.56(0.24)	0.49(0.25)
Fr15	0.48(0.26)	0.62(0.21)	0.58(0.24)	0.41(0.21)
XK10	0.38(0.24)	0.50(0.24)	0.48(0.26)	0.29(0.18)
BS15	0.55(0.25)	0.70(0.18)	0.63(0.23)	0.43(0.25)
BS-1NC	0.54(0.26)	0.65(0.20)	0.61(0.24)	0.42(0.26)
BS-1OC	0.46(0.28)	0.51(0.28)	0.52(0.24)	0.43(0.25)
BS-1QP	0.42(0.23)	0.53(0.23)	0.49(0.27)	0.36(0.22)
BS-2nF	0.46(0.22)	0.59(0.19)	0.53(0.22)	0.37(0.25)
BS-2CG	0.48(0.29)	0.64(0.22)	0.58(0.27)	0.33(0.28)
BS-2CM	0.55(0.25)	0.70(0.21)	0.62(0.22)	0.40(0.24)
BS-23M	0.62(0.24)	**0.73(0.17)**	0.63(0.22)	0.47(0.25)
BS-4OP	0.57(0.24)	0.64(0.20)	0.59(0.24)	0.39(0.27)
BS-4OS	**0.68(0.21)**	0.71(0.18)	**0.66(0.23)**	**0.63(0.26)**
BS-6PC	0.55(0.25)	0.71(0.17)	0.62(0.23)	0.43(0.26)
BS-6PS	0.55(0.26)	0.71(0.17)	0.64(0.23)	0.43(0.25)
BS-6PQ	0.38(0.24)	0.57(0.21)	0.50(0.21)	0.36(0.25)

For the systematically modified configurations, Table 11 shows the text location detection results, Table 12 the coverage assessment, and Table 13 shows the text recognition results. The best location detection F1-measure of 0.67 is achieved by *(BS-4OS)*, which is also supported by the coverage assessment in Table 12 with the highest F1-measure of 0.65. The separate evaluation of each dataset in Table 4 confirms as well that *(BS-4OS)* works best for EconBiz, CHIME-R, and CHIME-S with F1-measures of 0.68, 0.66, and 0.63. The best result for DeGruyter is 0.73 by *(BS-23M)* with *(BS-4OS)* having the second best F1-measure of 0.71. The best cov-

Table 5 Precision (Pr), Recall (Re), and F1-measure for the Average Text Element Coverage. Results are averaged over all datasets for configurations from the literature.

Config.	*Pr*	*Re*	*F*1 (*SD*)
SZ13	0.52	**0.59**	0.47 (0.21)
Hu05	**0.79**	0.54	**0.57 (0.20)**
Ja07	0.41	0.32	0.32 (0.21)
BS15	0.60	0.49	0.50 (0.24)
CK15	0.53	0.41	0.42 (0.21)
Fr15	0.65	0.54	0.54 (0.23)
XK10	0.33	0.34	0.30 (0.22)

erage assessment results for EconBiz (0.56), DeGruyter (0.67), CHIME-R (0.71), and CHIME-S (0.66) are by *(BS-4OS)*, as shown in Table 6. Configuration *(BS-4OS(Ocropy))* also produces the best text recognition results with an average local Levenshtein of 4.71 and an OPC of 0.53. Comparing the performance of the three OCR engines and different configurations on each dataset individually (see Table 8, Table 9, and Table 10), the best local Levenshtein is also achieved by configuration *(BS-4OS(Ocropy))* with values between 3.51 and 5.80. In addition, configuration *(BS-4OS(Ocropy))* shows the best results of 95.49 for the average global Levenshtein Distance. Comparing the different, systematically modified configurations per step of the pipeline shows that the only major improvement is achieved by *(BS-4OS)*. We have also evaluated most of the systematic configurations using a test license of the commercial OCR engine ABBYY FineReader. The text recognition results are shown in

Table 6 Average F1-measure and Standard Deviation for Text Element Coverage per configuration for the individual datasets EconBiz, DeGruyter, CHIME-R, and CHIME-S.

Config.	EconBiz	DeGruyter	CHIME-R	CHIME-S
SZ13	0.42(0.18)	0.44(0.23)	0.49(0.21)	0.55(0.20)
Hu05	0.48(0.17)	0.58(0.22)	0.66(0.18)	0.47(0.22)
Ja07	0.28(0.19)	0.33(0.20)	0.41(0.22)	0.21(0.19)
CK15	0.37(0.19)	0.48(0.20)	0.45(0.22)	0.35(0.23)
Fr15	0.42(0.21)	0.62(0.15)	0.62(0.25)	0.45(0.25)
XK10	0.23(0.17)	0.34(0.21)	0.37(0.26)	0.16(0.11)
BS15	0.40(0.20)	0.62(0.14)	0.60(0.26)	0.31(0.21)
BS-1NC	0.36(0.21)	0.58(0.16)	0.58(0.26)	0.27(0.17)
BS-1OC	0.32(0.24)	0.46(0.28)	0.42(0.25)	0.31(0.21)
BS-1QP	0.43(0.22)	0.45(0.23)	0.43(0.26)	0.33(0.22)
BS-2nF	0.40(0.17)	0.56(0.16)	0.60(0.22)	0.35(0.23)
BS-2CG	0.46(0.19)	0.58(0.16)	0.69(0.19)	0.45(0.23)
BS-2CM	0.36(0.20)	0.60(0.16)	0.59(0.25)	0.24(0.18)
BS-23M	0.39(0.19)	0.62(0.14)	0.56(0.22)	0.30(0.17)
BS-4OP	0.37(0.17)	0.47(0.14)	0.48(0.21)	0.20(0.20)
BS-4OS	**0.56(0.14)**	**0.67(0.11)**	**0.71(0.21)**	**0.66(0.20)**
BS-6PC	0.39(0.20)	0.62(0.15)	0.60(0.26)	0.30(0.20)
BS-6PS	0.39(0.20)	0.61(0.15)	0.59(0.26)	0.31(0.21)
BS-6PQ	0.21(0.16)	0.41(0.20)	0.36(0.22)	0.19(0.17)

Table 7 Average local Levenshtein (L) and global Levenshtein (G) and Operations Per Character (OPC) over all datasets for the configurations from the literature using Tesseract.

Config.	$AVG_L(SD)$	$AVG_G(SD)$	OPC
SZ13	6.67 (4.82)	122.28 (141.03)	0.70
Hu05	6.65 (5.41)	126.35 (138.95)	0.71
Ja07	7.92 (5.56)	150.25 (140.59)	1.13
BS15	6.23 (4.93)	**108.81 (108.53)**	**0.67**
CK15	**6.07 (5.08)**	120.12 (125.87)	0.71
Fr15	6.72 (6.02)	135.64 (201.31)	0.85
XK10	7.06 (5.41)	125.45 (134.88)	0.74

Table 14. The results vary in a range between the results of Tesseract and Ocropy and do not present a better configuration than *(BS-4OS)*.

Fig. 5 compares the average execution time over all figures from all four datasets for the seven configurations motivated from the literature. The results are shown in seconds at log scale, due to the varying runtime performance. As one can see from the figure, the configurations *(SZ13)*, *(Hu05)*, and *(Ja07)* are in the same range, with the latter one being the fastest configuration. The other configurations are by an order of magnitude slower, with *(CK15)* being the configuration needing most of the time (about 38 days for all four datasets). A detailed analysis of the data generated by the systematically modified configurations led to the following observations. The fastest binarization is Otsu's method with an average execution time of less than 100ms per figure (averaged over all figures from all datasets). The adaptive binarization requires the most time with about two minutes on average. The pivoting algorithm requires

Table 8 Average Levenshtein and Standard Deviation for Text Recognition Quality per configuration for the individual datasets EconBiz, DeGruyter, CHIME-R, and CHIME-S using Tesseract.

Config	EconBiz	DeGruyter	CHIME-R	CHIME-S
SZ13	6.00(3.24)	6.13(3.25)	7.10(6.25)	7.29(5.64)
Hu05	5.69(3.33)	5.69(3.74)	6.75(6.95)	7.94(6.13)
Ja07	7.15(3.62)	7.97(3.85)	7.94(7.25)	8.36(6.34)
CK15	5.27(3.11)	4.96(2.78)	6.65(7.21)	6.74(5.30)
Fr15	5.74(3.34)	6.53(5.87)	6.81(7.76)	7.26(5.55)
XK10	6.27(3.46)	6.46(3.56)	6.99(7.14)	7.88(6.09)
BS15	5.42(3.06)	4.88(2.58)	6.51(6.85)	7.21(5.34)
BS-1NC	5.47(3.12)	5.30(2.78)	6.42(6.85)	7.22(5.42)
BS-1OC	5.74(3.20)	5.76(3.05)	6.72(6.72)	7.51(5.71)
BS-1QP	7.13(4.13)	8.44(4.88)	8.50(7.88)	8.68(6.49)
BS-2nF	5.78(3.33)	5.83(3.11)	6.64(6.29)	7.63(5.72)
BS-2CG	5.77(3.43)	5.49(3.79)	6.63(7.46)	8.34(6.27)
BS-2CM	5.57(3.07)	4.95(2.75)	6.72(7.27)	7.51(6.08)
BS-23M	5.17(3.16)	4.92(3.34)	6.66(6.68)	7.53(5.62)
BS-4OP	7.56(3.95)	8.82(3.84)	8.15(7.25)	8.49(6.47)
BS-4OS	**4.90(3.01)**	**4.55(2.55)**	**5.76(6.01)**	**6.27(4.97)**
BS-6PC	5.13(3.03)	4.77(2.33)	6.06(6.71)	7.29(5.43)
BS-6PS	5.39(3.02)	4.83(2.44)	6.53(6.87)	7.24(5.37)
BS-6PQ	5.47(3.13)	4.96(2.63)	6.31(6.91)	7.43(5.70)

Table 9 Average Levenshtein and Standard Deviation for Text Recognition Quality per configuration for the individual datasets EconBiz, DeGruyter, CHIME-R, and CHIME-S using Ocropy.

Config.	EconBiz	DeGruyter	CHIME-R	CHIME-S
BS15	4.80(2.93)	4.07(2.26)	5.96(6.97)	7.20(5.66)
BS-1NC	4.81(2.86)	4.47(2.83)	5.94(7.00)	7.47(5.71)
BS-1OC	5.16(3.10)	5.21(3.23)	6.54(6.89)	7.35(5.89)
BS-1QP	6.18(3.33)	6.29(3.04)	7.19(7.72)	8.51(6.27)
BS-2nF	5.30(2.94)	5.02(2.58)	6.91(7.83)	7.78(6.22)
BS-2CG	5.28(3.27)	4.71(2.92)	6.31(7.83)	8.14(6.29)
BS-2CM	4.98(3.00)	4.45(2.42)	6.09(7.35)	7.73(6.10)
BS-23M	4.56(2.89)	3.82(2.24)	6.13(7.01)	7.33(5.60)
BS-4OP	6.34(3.29)	6.71(3.44)	7.36(7.81)	8.22(6.44)
BS-4OS	**3.86(2.82)**	**3.51(2.00)**	**5.08(6.32)**	**5.80(5.50)**
BS-6PC	4.73(2.93)	4.28(2.23)	5.73(6.82)	7.19(5.63)
BS-6PS	4.75(2.89)	4.08(2.27)	5.79(6.87)	7.16(5.61)
BS-6PQ	5.20(2.99)	4.64(2.77)	5.95(6.59)	7.57(6.05)

Table 10 Average Levenshtein and Standard Deviation for Text Recognition Quality per configuration for the individual datasets EconBiz, DeGruyter, CHIME-R, and CHIME-S using ABBYY FineReader OCR.

Config.	EconBiz	DeGruyter	CHIME-R	CHIME-S
BS15	5.44(3.08)	4.79(2.73)	6.19(6.71)	7.45(5.52)
BS-1NC	5.73(3.50)	5.09(3.03)	6.39(6.89)	7.28(5.57)
BS-1OC	5.89(3.48)	5.70(3.19)	6.86(6.46)	7.31(5.71)
BS-2nF	5.66(3.24)	5.31(3.08)	6.30(6.52)	7.36(5.69)
BS-2CG	5.99(3.48)	5.37(3.52)	6.78(7.76)	8.31(6.35)
BS-2CM	5.72(3.11)	4.77(2.44)	6.49(7.24)	7.93(6.22)
BS-23M	5.10(2.96)	4.57(2.90)	6.36(6.53)	7.66(5.82)
BS-4OP	7.80(3.98)	8.47(3.93)	8.10(7.95)	9.34(6.73)
BS-4OS	4.76(3.03)	4.30(2.30)	5.38(5.82)	5.09(4.68)
BS-6PC	4.71(2.96)	4.00(2.34)	5.36(6.63)	6.57(5.19)
BS-6PS	5.43(3.07)	4.75(2.68)	6.12(6.64)	7.43(5.49)
BS-6PQ	5.83(3.07)	5.77(2.69)	6.87(7.26)	7.85(5.96)

about twice as much time as Connected Component Labeling for region extraction. Regarding step 2 and 3 of the pipeline, only the morphological clustering differs a lot from the rest with an average execution time of several minutes or more. The PSD and Hough orientation estimation execute on average in a few milliseconds, while the SSOD needs several seconds. The comparison of Ocropy and Tesseract shows that the latter one is about three times faster. Finally, all post-processing methods need on average less than a millisecond.

6.2 Non-Linear Pipeline Configurations

In our experiments, configuration *(BS-4OS/R1)* resulted in similar results regarding the average F1-measures over all datasets as without the additional binarization step. However, the text recognition quality decreased with respect to the average global Levenshtein distance, the average local Levenshtein distance, as well as the opera-

Table 11 Precision (Pr), Recall (Re), and F1-measure for the Average Text Location Detection, Element Ratio (ER), and Matched Element Ratio (MER). Results are averaged over all datasets for the systematically modified configurations.

Config.	*Pr*	*Re*	*F*1 (*SD*)	*ER*	*MER*
BS15	0.66	0.55	0.58 (0.25)	1.04	0.69
BS-1NC	0.64	0.52	0.57 (0.25)	0.96	0.64
BS-1OC	0.67	0.40	0.49 (0.26)	0.74	0.53
BS-1QP	0.61	0.44	0.48 (0.25)	0.96	0.75
BS-2nF	0.60	0.46	0.51 (0.23)	0.86	0.52
BS-2CG	0.62	0.50	0.55 (0.27)	0.90	0.64
BS-2CM	0.61	0.54	0.59 (0.25)	1.19	0.74
BS-23M	0.67	0.55	0.62 (0.23)	1.08	0.65
BS-4OP	0.62	0.53	0.57 (0.24)	**1.01**	0.66
BS-4OS	0.67	**0.63**	**0.67 (0.22)**	1.27	**0.88**
BS-6PC	**0.69**	0.54	0.59 (0.25)	0.97	0.70
BS-6PS	0.67	0.55	0.60 (0.25)	**1.01**	0.69
BS-6PQ	0.66	0.38	0.48 (0.25)	0.60	0.43

tions per character. The comparison with the results of the non-linear configuration with repetition of the full pipeline *(BS-4OS/R2)* shows worse results than with the linear configuration. Table 15 lists the detailed results of the comparison of the linear and non-linear configurations. Both non-linear pipelines are based on the linear configuration *(BS-4OS-O)* and execute the first step, which takes the most time, several times or even repeats the whole pipeline. Thus, the executions of the non-linear configuration pipelines take much longer than their linear configuration counterpart.

6.3 Two-Pass Pipeline Configurations

Two different configurations were tested for the two-pass approach. Configuration *(BS-4OS-TP-0avg)* uses a threshold of 0.0 for the average confidence value of the text line,

Table 12 Precision (Pr), Recall (Re), and F1-measure for the Average Text Element Coverage. Results are averaged over all datasets for the systematically modified configurations.

Config.	*Pr*	*Re*	*F*1 (*SD*)
BS15	0.60	0.49	0.50 (0.24)
BS-1NC	0.59	0.44	0.47 (0.24)
BS-1OC	0.46	0.40	0.38 (0.26)
BS-1QP	0.41	0.57	0.42 (0.23)
BS-2nF	0.59	0.54	0.50 (0.21)
BS-2CG	0.76	0.54	0.57 (0.20)
BS-2CM	0.57	0.47	0.47 (0.24)
BS-23M	0.60	0.47	0.48 (0.22)
BS-4OP	0.49	0.40	0.41 (0.20)
BS-4OS	**0.77**	**0.63**	**0.65 (0.17)**
BS-6PC	0.59	0.49	0.49 (0.24)
BS-6PS	0.59	0.49	0.49 (0.24)
BS-6PQ	0.39	0.29	0.31 (0.21)

Table 13 Average local Levenshtein (L) and global Levenshtein (G) and Operations Per Character (OPC) over all datasets for the systematic configurations.

	Tesseract			Ocropy		
Config.	$AVG_L(SD)$	$AVG_G(SD)$	OPC	$AVG_L(SD)$	$AVG_G(SD)$	OPC
BS15	6.23 (4.93)	108.81 (108.53)	0.67	5.47 (4.98)	108.55 (106.64)	0.64
BS-1NC	6.27 (4.95)	117.58 (124.23)	0.69	5.70 (5.09)	117.46 (128.73)	0.66
BS-1OC	6.55 (5.06)	131.58 (142.74)	0.75	6.16 (5.21)	131.39 (143.16)	0.73
BS-1QP	8.31 (6.14)	154.54 (168.10)	1.09	7.06 (5.62)	136.40 (132.05)	0.82
BS-2nF	6.55 (4.94)	111.30 (105.13)	0.75	6.29 (5.50)	120.71 (109.18)	0.76
BS-2CG	6.68 (5.65)	108.86 (102.93)	0.66	6.22 (5.75)	130.21 (127.87)	0.69
BS-2CM	6.30 (5.29)	115.43 (113.79)	0.69	5.85 (5.34)	110.74 (107.23)	0.67
BS-23M	6.15 (5.12)	104.61 (105.97)	0.63	5.52 (5.10)	106.71 (104.05)	0.64
BS-4OP	8.30 (5.59)	147.91 (129.55)	1.04	7.23 (5.60)	135.21 (122.48)	0.85
BS-4OS	5.47 (4.39)	96.29 (99.44)	0.58	4.71 (4.66)	95.49 (94.80)	0.53
BS-6PC	5.96 (4.88)	105.50 (107.16)	0.61	5.46 (5.00)	109.07 (104.57)	0.63
BS-6PS	6.20 (4.90)	108.06 (109.38)	0.64	5.45 (4.96)	106.38 (103.29)	0.63
BS-6PQ	6.07 (5.03)	120.78 (122.44)	0.67	5.79 (4.97)	126.92 (124.06)	0.71

Table 14 Average local Levenshtein (L) and global Levenshtein (G) and Operations Per Character (OPC) over all datasets for the systematic configurations with ABBYY FineReader OCR.

Config.	$AVG_L(SD)$	$AVG_G(SD)$	OPC
BS15	6.08 (4.94)	119.47 (109.53)	0.76
BS-1NC	6.28 (5.12)	124.91 (125.83)	0.78
BS-1OC	6.54 (4.94)	137.37 (141.49)	0.85
BS-2nF	6.19 (4.91)	129.07 (125.91)	0.86
BS-2CG	6.75 (5.74)	105.06 (86.48)	0.64
BS-2CM	6.22 (5.25)	123.97 (111.51)	0.78
BS-23M	6.06 (5.04)	111.50 (94.13)	0.75
BS-4OP	8.45 (5.94)	156.56 (132.74)	1.08
BS-4OS	5.08 (4.32)	107.18 (106.71)	0.63
BS-6PC	5.19 (4.69)	95.39 (94.47)	0.53
BS-6PS	6.05 (4.91)	119.67 (107.84)	0.73
BS-6PQ	6.58 (5.06)	142.36 (134.57)	0.80

Table 15 F1-measure ($F1_D$) for Text Location Detection, F1-measure ($F1_C$) for Text Element Coverage, Average Local Levenshtein (AVG_L), Average Global Levenshtein (AVG_G) and operations per character (OPC) for the linear baseline *(BS-4OS)* and the non-linear configurations *(BS-4OS/R1)* and *(BS-4OS/R2)*.

Config.	$F1_D(SD)$	$F1_C(SD)$	$AVG_L(SD)$	AVG_G	OPC
BS-4OS (Tesseract)	0.67 (0.22)	0.65 (0.17)	5.47 (4.39)	96.29	0.58
BS-4OS (Ocropy)	0.64 (0.21)	0.55 (0.20)	4.71 (4.66)	95.49	0.53
BS-4OS/R1 (Tesseract)	0.67 (0.22)	0.65 (0.18)	5.64 (4.80)	102.72	0.60
BS-4OS/R1 (Ocropy)	0.65 (0.21)	0.55 (0.21)	4.78 (4.66)	96.30	0.55
BS-4OS/R2 (Tesseract)	0.62 (0.22)	0.63 (0.17)	5.86 (4.97)	106.08	0.62
BS-4OS/R2 (Ocropy)	0.60 (0.22)	0.51 (0.20)	4.82 (4.62)	97.28	0.55

whereas the configuration *(BS-4OS-TP-0min)* accepts only text lines with a minimum confidence of 0.0. Both configurations improved the results of the best linear pipeline configuration *(BS-4OS)*. Regarding the $F1_D$-measure both two-pass configurations

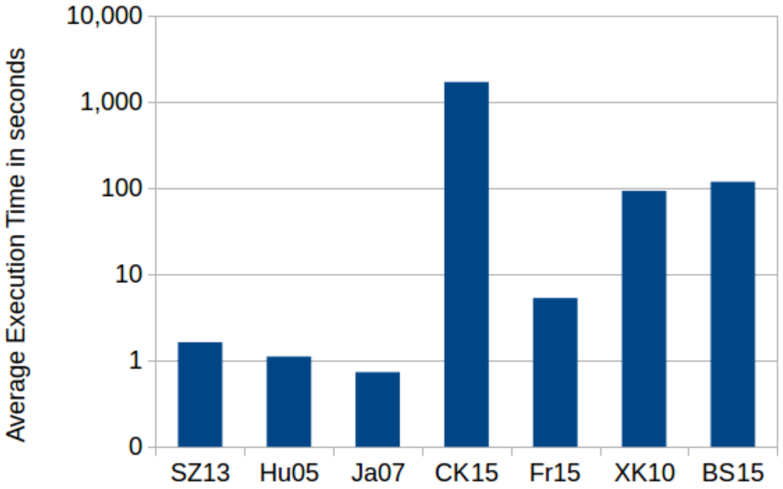

Fig. 5 Average execution time for the configurations motivated from the literature

improved the previous best results by 0.02. The text recognition quality clearly improved with respect to the average global Levenshtein distance, the average local Levenshtein distance, and the operations per character. Configuration *(BS-4OS-TP-0min)* achieved the overall best results. Table 16 lists the detailed results of the two-pass configurations compared with the linear pipeline configuration *(BS-4OS)*.

Table 16 F1-measure ($F1_D$) for Text Location Detection, F1-measure ($F1_C$) for Text Element Coverage, Average Local Levenshtein (AVG_L), Average Global Levenshtein (AVG_G) and operations per character (OPC) for the linear baseline *(BS-4OS)* and the two-pass configurations *(BS-4OS-TP-0avg)* and *(BS-4OS-TP-0min)*.

Config.	$F1_D(SD)$	$F1_C(SD)$	$AVG_L(SD)$	AVG_G	OPC
BS-4OS (Tesseract)	0.67 (0.22)	0.65 (0.17)	5.47 (4.39)	96.29	0.58
BS-4OS (Ocropy)	0.64 (0.21)	0.55 (0.20)	4.71 (4.66)	95.49	0.53
BS-4OS-TP-0avg	0.69 (0.20)	0.65 (0.16)	3.89 (4.64)	83.09	0.36
BS-4OS-TP-0min	0.69 (0.20)	0.64 (0.16)	3.59 (4.26)	77.74	0.35

The performance measures for the two-pass configuration showed that the initial OCR step increases the runtime performance only marginally and can be neglected. Thus, the two-pass configuration is the overall best configuration with respect to the extraction quality and runtime performance.

7 Discussion

7.1 Linear Pipeline Configurations

Comparing the different linear configurations from the literature shows that our TX pipeline *(BS15)* works best. A possible reason is that our pipeline does not make many assumptions about the figures, e. g. figure type, font, or color. Thus, performing better on the heterogeneous datasets. In the following, we will discuss the results for the individual pipeline steps based on the results from the systematically modified linear configurations. Comparing the configurations for the first pipeline step leads to the conclusion that the adaptive binarization works best because it can adapt to local color variations in a figure. Otsu's method is too simple and Niblack's method is more suited for document images which have fewer color variations. The non-competitive results for the pivoting algorithm can be explained with the larger regions and the possibility that a region can be a mixture of text and graphic elements due to the only horizontal and vertical subdivision. Looking at step 2 and 3 of the pipeline, only the morphological clustering shows slightly better results than the DBSCAN-MST combination, most likely due to its processing on pixel level. When comparing the different orientation calculation methods in step 4, the Single String Orientation Detection works best. One explanation can be that the orientation estimation via Hough in the base configuration works on the centers of mass of character regions, which is an aggregated region representation, while the SSOD in *(BS-4OS)* computes the orientation on the original pixels. Thus, it avoids a possible error, induced by the pixel aggregation. When comparing the OCR engines from step 5, Ocropy generally produces better results than Tesseract. Ocropy seems to be more conservative, having built in much more restrictions about what input to accept and when to execute the OCR. Furthermore, each OCR engine comes with its own English language model and we did not evaluate their influence. Finally, if we compare the open source OCR engines Tesseract and Ocropy with the commercial ABBYY FineReader OCR within in the linear pipeline, we can see that the FineReader's text recognition results are in the same range as the open source engines. This leads to the conclusion, that the OCR step does not have that much influence on the overall performance of the text extraction pipeline. The methods for post-processing do not improve the results. One reason might be the simplicity of the methods, the other that the later steps of the pipeline have less influence on the overall results, as can be seen in the OCR step comparison. The best results regarding linear configurations were achieved by *(BS-4OS)* as presented in our earlier work [3].

The runtime performance results for the configurations motivated from the literature clearly show that the first pipeline step is the most expensive one with respect to the execution time. Those configurations which use standard Otsu's method are a lot faster than those using Adaptive Otsu Binarization. This difference was expected since the latter one recursively processes a figure while applying Otsu's method multiple times. In addition, configuration *(CK15)* did not finish in reasonable time (about 2 hours per image, 38 days in total) due to its expensive morphological clustering. This method is very expensive since it iterates multiple times over an image increasing the region by single pixel borders each time. The systematic comparisons confirm these results. Furthermore, it shows that the SSOD is more expensive than Hough and PSD because it works on the pixel level similar to the morphological clustering. The difference between Ocropy and Tesseract can be explained by the fact that Tesseract is used via an API while Ocropy is currently used via command line which requires system calls. The fast execution of the post-processing methods results from their simplicity, applying only simple string comparisons.

7.2 Non-Linear Pipeline Configurations

The results for the non-linear configurations support our previous discovery that a modification in or after the OCR processing step cannot improve the results because the information is already lost at this point. Thus, the small improvement by repeating the binarization is most likely a random deviation. The results also conform with the text recognition results with the commercial ABBYY FineReader, which differed only slightly (see Table 14). This means that in the region extraction, region classification, text line detection, or orientation estimation steps before the text recognition, important information is lost so that the OCR engines are not able to produce better recognition results from the given input. This explains also why a repetition of the pipeline after the text recognition step does not improve the results.

7.3 Two-Pass Pipeline Configurations

Our preliminary experiments in Section 4.3 suggested that the combination of ABBYY FineReader as OCR engine to detect horizontal text and the linear configuration *(BS-4OS)* to recognize rotated text could improve the recognition results. The evaluation confirmed that the two-pass extension of the best configuration *(BS-4OS(Ocropy))* improves the results even further. However, one difficulty is the choice of an appropriate decision criterion to accept the output of ABBYY FineReader OCR. Using the minimum confidence value as criteria produced higher quality regarding average global and local Levenshtein distances. However, the results regarding F1-measures for text location detection and text element coverage were not improved compared with the configuration using the average confidence value. This means that less output was accepted and removed from the figure but not more text was correctly recognized. One explanation might be that the linear configuration was not able to perform better on text which contained words with low confidence values. This might be due to bad image quality or characters/symbols that are unknown to the OCR engines and thus, cannot be recognized. Another possible reason, which we did not investigate further, is the modification of the image before applying the linear pipeline, which could have introduced additional noise since only the color white is used to erase the text that was already extracted.

8 Conclusion

Starting with the 32 linear pipeline configurations for text extraction, our comparison [3] showed that there is a clear favorite configuration *(BS-4OS(Ocropy))*. We extended our set of configurations by introducing the commercial ABBYY FineReader OCR. However, the evaluation of the new configurations did not show a significant improvement of the recognition results. The results of the non-linear configurations for text extraction did not improve the results of the linear configurations either. Thus, we can conclude that valuable information is lost before applying the OCR. The developed two-pass approach achieved the best results with configuration *(BS-4OS-TP-0min)* which uses the commercial OCR engine ABBYY FineReader to extract horizontal text and then uses the best linear configuration *(BS-4OS(Ocropy))* to extract rotated text. Concluding from the results of our experiments, a more detailed investigation of the region extraction and region classification steps is needed to further improve the results. Further extensions to handle subscripts, superscripts, and mathematical formulas should be investigated as well. However, this requires a more precise, multi-level gold standard which, to the best of our knowledge, does not exist so far. Finally, as stated in the introduction, we are providing the datasets and the implementation of the generic linear pipeline that was used in our experiment to the public. This allows for integrating and comparing new methods as well as the reproduction of our results.

References

1. F. Böschen and A. Scherp. Formalization and preliminary evaluation of a pipeline for text extraction from infographics. In R. Bergmann, S. Görg, and G. Müller, editors, *Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB, Trier, Germany, October 7-9, 2015.*, volume 1458 of *CEUR Workshop Proceedings*, pages 20–31. CEUR-WS.org, 2015.
2. F. Böschen and A. Scherp. Multi-oriented text extraction from information graphics. In C. Vanoirbeek and P. Genevès, editors, *Proceedings of the 2015 ACM Symposium on Document Engineering, DocEng 2015, Lausanne, Switzerland, September 8-11, 2015*, pages 35–38. ACM, 2015.
3. F. Böschen and A. Scherp. A comparison of approaches for automated text extraction from scholarly figures. In *MultiMedia Modeling - 23rd International Conference, MMM 2017, Reykjavik, Iceland, January 4-6, 2017, Proceedings, Part I*, volume 10132 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2017.
4. S. Carberry, S. Elzer, and S. Demir. Information graphics: an untapped resource for digital libraries. In E. N. Efthimiadis, S. T. Dumais, D. Hawking, and K. Järvelin, editors, *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, pages 581–588. ACM, 2006.
5. S. Carberry, S. E. Schwartz, K. F. McCoy, S. Demir, P. Wu, C. F. Greenbacker, D. Chester, E. Schwartz, D. Oliver, and P. S. Moraes. Access to multimodal articles for individuals with sight impairments. *ACM Trans. Interact. Intell. Syst.*, 2(4):21, 2012.
6. Z. Chen, M. J. Cafarella, and E. Adar. Diagramflyer: A search engine for data-driven diagrams. In A. Gangemi, S. Leonardi, and A. Panconesi, editors, *Proceedings of the 24th International Conference on World Wide Web Companion, WWW 2015, Florence, Italy, May 18-22, 2015 - Companion Volume*, pages 183–186. ACM, 2015.
7. D. Chester and S. Elzer. Getting computers to see information graphics so users do not have to. In M. Hacid, N. V. Murray, Z. W. Ras, and S. Tsumoto, editors, *Foundations of Intelligent Systems, 15th International Symposium, ISMIS 2005, Saratoga Springs, NY, USA, May 25-28, 2005, Proceedings*, volume 3488 of *Lecture Notes in Computer Science*, pages 660–668. Springer, 2005.

8. Y. Chiang and C. A. Knoblock. A general approach for extracting road vector data from raster maps. *International Journal on Document Analysis and Recognition (IJDAR)*, 16(1):55–81, 2013.
9. Y. Chiang and C. A. Knoblock. Recognizing text in raster maps. *GeoInformatica*, 19(1):1–27, 2015.
10. S. R. Choudhury and C. L. Giles. An architecture for information extraction from figures in digital libraries. In A. Gangemi, S. Leonardi, and A. Panconesi, editors, *Proceedings of the 24th International Conference on World Wide Web Companion, WWW 2015, Florence, Italy, May 18-22, 2015 - Companion Volume*, pages 667–672. ACM, 2015.
11. M. P. Deseilligny, H. L. Men, and G. Stamon. Character string recognition on maps, a rotation-invariant recognition method. *Pattern Recognition Letters*, 16(12):1297–1310, 1995.
12. M. Fraz, M. S. Sarfraz, and E. A. Edirisinghe. Exploiting colour information for better scene text detection and recognition. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(2):153–167, 2015.
13. G. Gao, H. Zhang, and H. Chen. A robust video text extraction and recognition approach using OCR feedback information. In Y. Ho, J. Sang, Y. M. Ro, J. Kim, and F. Wu, editors, *Advances in Multimedia Information Processing - PCM 2015 - 16th Pacific-Rim Conference on Multimedia, Gwangju, South Korea, September 16-18, 2015, Proceedings, Part I*, volume 9314 of *Lecture Notes in Computer Science*, pages 507–517. Springer, 2015.
14. J. Gllavata and B. Freisleben. Adaptive fuzzy text segmentation in images with complex backgrounds using color and texture. In A. Gagalowicz and W. Philips, editors, *Computer Analysis of Images and Patterns, 11th International Conference, CAIP 2005, Versailles, France, September 5-8, 2005, Proceedings*, volume 3691 of *Lecture Notes in Computer Science*, pages 756–765. Springer, 2005.
15. W. Huang and C. L. Tan. A system for understanding imaged infographics and its applications. In P. R. King and S. J. Simske, editors, *Proceedings of the 2007 ACM Symposium on Document Engineering, Winnipeg, Manitoba, Canada, August 28-31, 2007*, pages 9–18. ACM, 2007.
16. W. Huang, C. L. Tan, and W. K. Leow. Associating text and graphics for scientific chart understanding. In *Eighth International Conference on Document Analysis and Recognition (ICDAR 2005), 29 August - 1 September 2005, Seoul, Korea*, pages 580–584. IEEE Computer Society, 2005.
17. J. Illingworth and J. Kittler. A survey of the hough transform. *Computer Vision, Graphics, and Image Processing*, 44(1):87–116, 1988.
18. C. Jayant, M. Renzelmann, D. Wen, S. Krisnandi, R. E. Ladner, and D. Comden. Automated tactile graphics translation: in the field. In E. Pontelli and S. Trewin, editors, *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility, ASSETS 2007, Tempe, Arizona, USA, October 15-17, 2007*, pages 75–82. ACM, 2007.
19. Z. Jiuzhou. Creation of synthetic chart image database with ground truth. Honors year project report, National University of Singapore, 2006. `https://www.comp.nus.edu.sg/~tancl/ChartImageDatabase/Report_Zhaojiuzhou.pdf`.
20. K. Khurshid, I. Siddiqi, C. Faure, and N. Vincent. Comparison of Niblack inspired binarization methods for ancient documents. In K. Berkner and L. Likforman-Sulem, editors, *Document Recognition and Retrieval XVI, DRR 2009, 16th Document Recognition and Retrieval Conference, part of the IS&T-SPIE Electronic Imaging Symposium, San Jose, CA, USA, January 18-22, 2009. Proceedings*, volume 7247 of *SPIE Proceedings*, pages 1–10. SPIE, 2009.
21. S. Lu, T. Chen, S. Tian, J. Lim, and C. L. Tan. Scene text extraction based on edges and support vector regression. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(2):125–135, 2015.
22. X. Lu, S. Kataria, W. J. Brouwer, J. Z. Wang, P. Mitra, and C. L. Giles. Automated analysis of images in documents for intelligent document search. *International Journal on Document Analysis and Recognition (IJDAR)*, 12(2):65–81, Jul 2009.
23. J. I. Olszewska. Active contour based optical character recognition for automated scene understanding. *Neurocomputing*, 161:65–71, 2015.
24. N. Otsu. A threshold selection method from gray-level histograms. *Systems, Man and Cybernetics, IEEE Transactions on*, 9(1):62–66, Jan 1979.
25. H. Samet and M. Tamminen. Efficient component labeling of images of arbitrary dimension represented by linear bintrees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(4):579–586, 1988.
26. J. Sas and A. Zolnierek. Three-stage method of text region extraction from diagram raster images. In R. Burduk, K. Jackowski, M. Kurzynski, M. Wozniak, and A. Zolnierek, editors, *Proceedings of the 8th International Conference on Computer Recognition Systems CORES 2013, Milkow, Poland, 27-29 May 2013*, volume 226 of *Advances in Intelligent Systems and Computing*, pages 527–538. Springer, 2013.

27. M. Savva, N. Kong, A. Chhajta, F. Li, M. Agrawala, and J. Heer. Revision: automated classification, analysis and redesign of chart images. In J. S. Pierce, M. Agrawala, and S. R. Klemmer, editors, *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, Santa Barbara, CA, USA, October 16-19, 2011*, pages 393–402. ACM, 2011.
28. C. M. Strohmaier, C. Ringlstetter, K. U. Schulz, and S. Mihov. Lexical postcorrection of ocr-results: The web as a dynamic secondary dictionary? In *7th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3-6 August 2003, Edinburgh, Scotland, UK*, pages 1133–1137. IEEE Computer Society, 2003.
29. S. Xu and M. Krauthammer. A new pivoting and iterative text detection algorithm for biomedical images. *Journal of Biomedical Informatics*, 43:924–931, 2010.
30. L. Yang, W. Huang, and C. L. Tan. Semi-automatic ground truth generation for chart image recognition. In H. Bunke and A. L. Spitz, editors, *Document Analysis Systems VII, 7th International Workshop, DAS 2006, Nelson, New Zealand, February 13-15, 2006, Proceedings*, volume 3872 of *Lecture Notes in Computer Science*, pages 324–335. Springer, 2006.