

The Hebrew University of Jerusalem
Faculty of Sciences
School of Computer Science and Engineering

Handwriting Recognition and Fast Retrieval
for Hebrew Historical Manuscripts

Shahar Armon

Master's Thesis
supervised by Prof. Michael Werman

December 2011

תקציר

תזה זו מתארת מערכת חדשה לזיהוי טקסט וחיפוש בכתבי יד עבריים היסטוריים. לאחר עיבוד של תמונות כתבי היד המערכת מחשבת ביעילות את הקריאה הסבירה ביותר, מציעה קריאות חלופיות (בעזרת כלי אינטראקטיבי) ובנוסף, מאפשרת לחפש מחרוזות בקריאות החלופיות של אוסף כתבי יד גדול.

כתבי יד היסטוריים מאופיינים בפגמים רבים וכן בגמישות של צורת האותיות בכתבי יד, לכן ישנם קשיים רבים בעיבוד וזיהוי מסמכים אלו. המערכת המתוארת מתמודדת עם הקושי ליצור סגמנטציה של אותיות (שיכולות להיות מחוברות או שבורות, או גם וגם) ועם הקושי בזיהוי אותיות (עקב מצבם הרעוע קשה לזהותן בוודאות). המערכת מפרידה אותיות מחוברות בשיטה חדשה על ידי חלוקת האותיות המחוברות לכמה חלקים. בהמשך, המערכת בודקת צירופים של חלקי אותיות (שנוצרו בעיבוד המקדים לשם ההפרדה או שהיו כבר מופרדים בתמונה) והצירופים הטובים מסווגים. שיטה משופרת שימשה לתיאור תמונת האותיות לשם הסיווג. כל האותיות שנמצאו סבירות וציוןן (שהתקבל מהסיווג) מאוכסנים בגרף הקריאה השומר גם את קשרי השכנות של האותיות. גרף הקריאה הינו מבנה הדומה ל- candidate-lattice והוא מתאר את הקריאות האלטרנטיבות של תמונת הטקסט.

בעזרת גרף הקריאה של תמונת טקסט ועל סמך הציונים שבו (הכוללים גם הסתברות של זוגות האותיות) המערכת מחשבת את הקריאה הסבירה ביותר. בנוסף, המערכת מאפשרת לבדוק את הקריאה שהתקבלה בעזרת כלי אינטראקטיבי. אותיות עם חלופות טובות מעכבות את מהלך הבדיקה וניתנת למשתמש אפשרות החלפה עבורן. לבסוף, המערכת מאכסנת את כל גרפי הקריאה במבנה נתונים מבוסס אינדקסים לשם שליפה מהירה על סמך חיפוש מחרוזות. באופן זה, אנו מאפשרים חיפוש מהיר בקריאות החלופיות של אוסף כתבי יד גדול.

תוצאות של ניסויים שערכנו על מסמכים מגניזת קהיר מראים שהמערכת מצליחה לקרא כתבי יד רבים הסובלים מבעיות מגוונות (הדבקות אותיות, אותיות שבורות, רעש, כתמים ועוד). בנוסף, המערכת מסוגלת לחפש בהצלחה ובמהירות רבה מחרוזות באוסף גדול של כתבי יד כאלו.

Abstract

In this thesis we describe a new system for handwriting recognition and fast retrieval of Hebrew historical manuscripts. After a processing stage, the system efficiently finds a plausible reading, offers alternative readings (via an interactive tool), and allows to search for strings in every alternative reading of every manuscript in a large collection.

The system faces the problems of under-segmentation (e.g. touching characters) and over-segmentation (e.g. broken characters). We present a novel method for over-segmenting touching characters. Then, in order to recognize letters in the image, candidates letters are generated from an over-segmented image by an improved method of feature-extraction. Letter candidates and their score are stored in a reading graph, which resembles a candidate-lattice, that captures alternative readings of the text image.

Experimental results on the Cairo Genizah show that our system successfully reads degraded manuscripts containing many abnormal letters (degraded, disconnected and touching), as well as retrieves manuscripts even when the image of the query string is degraded.

Acknowledgements

I wish to thank my advisor Prof. Michael Werman for his guidance and patience throughout the process of researching and writing this thesis.

I would like to thank Friedberg Genizah Project and to the team behind this wonderful project.

I also would like to thank my family for the help and the support they provided me through my entire life.

Special thanks to the friends at the school of computer science at the Hebrew University and especially to Shaul Almagor, for his great help along the academic way and with writing this thesis.

Contents

1	Introduction	2
2	Related work	4
2.1	OCR systems	4
2.2	Feature extraction and classification	4
2.3	Segmentation	5
2.4	Ambiguities in segmentation, and linguistic knowledge	6
2.5	Strings search	7
2.6	The Hebrew language	8
3	Document preprocessing	10
3.1	General preprocessing	10
3.2	Creating over-segmentation	11
4	Reading-graph model	14
4.1	Reading-graph construction	15
4.1.1	Join recognition	16
4.1.2	Updating the reading graph	18
4.2	Reading-graph applications	20
4.2.1	Finding the optimal reading	20
4.2.2	Interactive checking and correcting of output text	21
4.2.3	Efficient storage and fast retrieval	22
5	Experimental results	26
5.1	Feature extraction	26
5.2	Handwriting recognition	28
5.3	String retrieval	29
6	Summary	31
	Bibliography	32
A	Algorithm for Reading-graph construction	36
B	Compute path with maximum average	38

1 Introduction

The Cairo Genizah is huge collection of Hebrew degraded documents, collected between the 11th and the 19th centuries in Cairo, Egypt. Research on the Genizah has progressed very slowly due to problems of accessibility. In recent years Friedberg Genizah Project has been publishing the manuscripts on-line (www.genizah.org). Conversion of the manuscripts to digital text and retrieval of manuscripts by query can significantly contribute to the Genizah research.

In this thesis we describe our system for reading degraded historical manuscripts (as presened in Figure 1) and retrieving manuscripts based on string-querie, even when the strings are ambiguous in the manuscript. The system successfully reads many of the Genizah manuscripts and makes it possible to search for strings within them.

OCR (optical character recognition) systems are usually developed for a specific language and either for printed or handwritten documents. There are several relevant approaches for OCR [1].

The earliest and simplest approach is *template matching*, which is computationally expensive and not robust. A more common approach is the *statistical approach* [2], in which a segmented character image is represented as a feature-vector in a fixed dimensional space. Another approach is the *syntactic approach* that represent characters based on structure of primitives (parts of characters) and the interrelationships between them.

Most OCR systems, based on the statistical approach, include four steps: preprocessing, segmentation, feature extraction and classification. *Preprocessing* includes image enhancement and binarization, noise removal, skewness correction, and extraction of text lines. *Segmentation* is the step in which the image is divided into characters (or other units). *Feature extraction* involves transforming the character image (or other unit) into a feature vector. The last step, *classification*



Figure 1: An examples of a text images from the Genizah that contain touching letters and disconnected letters.

is where feature vectors are classified as specific characters.

After more than forty years of research OCR systems for machine-printed text show impressive performance [3]. However, the recognition of handwriting is still a central research field due to the inherent variance in handwriting. There are two types of OCR for handwriting. *On-line handwriting recognition* where the positions of the writer's pen is extra data. *Off-line handwriting recognition* only which analyzes only images.

One of the challenges in developing handwriting recognition for historical manuscripts is the low quality that is typical to historical manuscripts which were damaged by physical deterioration and environmental effects. This challenge is reflected in all of the four steps mentioned above.

This thesis presents a system for handwriting recognition and fast retrieval of Hebrew historical manuscripts written during the middle ages. The system deals with over-segmentation (e.g. broken characters) and under-segmentation (e.g. touching characters), and returns possible readings. The innovations of this thesis are mainly in the dealing with unrecognized segments and in focusing on several possible readings as even expert human readers often argue over the exact reading. In addition, this thesis presents improved methods to perform feature extraction, over-segmentation and fast retrieval.

The thesis is organized as follows. In Section 2 we review related work. In Section 3 we introduce the document preprocessing and the over-segmentation method we developed. In Section 4 we describe in detail our model, the conversion of the image to our model, and three applications of this model we developed. Section 5 gives experimental results, and finally, we summarize this thesis in Section 6.

2 Related work

Current OCR work deals mainly with handwriting. More than a decade ago [4] published a comprehensive survey on on-line and off-line handwriting recognition and new papers continue to be published (e.g. [5, 6]). There are also several full OCR systems for historical manuscripts, as reviewed in the next subsection. In addition, there is work that tries to deal with specific challenges relevant to historical manuscripts: feature extraction and classification methods (2.2); segmentation (2.3); ambiguities in segmentation, linguistic knowledge (2.4); and string search (2.5). At the end of this section we review properties of the Hebrew language and related work about Hebrew document analysis (2.6).

2.1 OCR systems

In [7], an approach is presented for the recognition of early Christian Greek manuscripts based on the detection of open and closed skeleton cavities. In the manuscripts there is no space between consecutive words, so a segmentation-free technique was used. In [8], an OCR methodology for recognizing historical Greek manuscripts, without knowing the font, is presented. It consists of a clustering scheme in order to group characters of similar shape and a manual step where labels are assigned to the clusters. Finally, an SVM is exploited for character classification. In [9] a character recognition system is presented, able to handle degraded manuscript documents. In contrast to the common approach no binarization preprocessing is needed. Instead, a modern object recognition methodology is adapted for the recognition: Interest points and local descriptors are extracted and then clustered in order to localize characters. Finally, character classification is based upon the localized characters.

2.2 Feature extraction and classification

One of the basic parts of OCR systems is the classification of character images. The classification is usually done by representing the character image as a vector of features and classifying the vector with a supervised learning algorithm. There are many feature extraction methods proposed for isolated characters. Some of the methods are based upon: moments [10], projections, crossings, profiles [11], contours or chain codes [12] curvature, thinning [7], etc. In addition to these

methods, there is a successful approach based upon recursive subdivisions of the character image [13, 14, 15, 16]. In practice, it is common to combine different methods possibly together with feature selection and dimension reduction.

The classification of the features is done by using common classification methods: statistical methods, kernel methods, artificial neural networks, etc. [17] presents an overview of classification methods and a comparison. Support vector machines (SVM) with radial basis function as the kernel often gives the highest accuracy, this is also shown in [18, 19].

2.3 Segmentation

The feature extraction methods presented previously are primarily designed for isolated character recognition. In practice, characters are not necessarily isolated, and therefore another challenge is to segment the image so each character will be one segment.

Under-segmentation is the main difficulty in character segmentation. There are several possible reasons in handwriting: cursive writing, dense writing, ink-spread, a writer's mistake, bad storage conditions and environmental effects. A common approach for dealing with this challenge is *explicit segmentation* [20]. In this approach, touching characters are segmented into isolated characters. Heuristics play a major role in explicit segmentation. In general, for every type of touching characters there might be a need for a different method based upon the nature of the relevant script and language (e.g. [21, 22, 10]). An example of explicit segmentation are the drop-fall algorithms which attempt to build a segmentation path by simulating a drop falling between two touching characters. This class of algorithms was first proposed by [23] and still appears and evolves in publications, e.g. [10, 24].

Over-segmentation is an additional difficulty in character segmentation that arises for example from characters that are broken into several segments. A common solution is to join such small segments into a single segment. Morphological operations might join such segments when they are very close [25, 26], however those operations can be destructive. Another solution is to join only small segments to the nearest character [8], but obviously this is not a complete solution.

Another approach, called the *holistic approach*, is to process units larger than characters in order to recognize whole words [27]. Another approach is

segmentation-free. In this approach global operators such as cross-correlation or finding interest points are applied to the image. Finally, the characters are recognized by processing the operators' results [7, 9].

2.4 Ambiguities in segmentation, and linguistic knowledge

Explicit-segmentation methods segment touching characters into isolated characters. There are problems with this approach. For example, the two English letters 'r' and 'n' may resemble the letter 'm' when adjacent. In such cases, several possible segmentations may be acceptable. Indeed, even an expert reader might not be able to interpret the text correctly. Degraded manuscripts contain irregular touching characters, character parts and noise. Therefore, in such documents, ambiguities in segmentation of isolated characters are very common.

An over-segmentation of the image can help resolve ambiguities. Then determining the character is combined with recognition. An over-segmentation can be represented by a graph, called a *candidate lattice* [28]. In this lattice, edges represent candidate characters, where a candidate character is a set of consecutive segments in the over-segmentation. A vertex represents the beginning of a character. Each candidate character is recognized by a classifier. Each path in the lattice represents possible segmentations of isolated characters. In [28], the most plausible path is found by a dynamic-programming algorithm and linguistic knowledge is used to verify this path. There are several variants of this approach. For example, in [29], beam search is used, and in [30], the authors test various language models combined with path evaluation in a lattice.

The incorporation of a language model is important due to the ambiguities in segmentation and recognition. For example, in the text 'my rnother' it is clear from context that the beginning of the second word should be 'm' and not 'r', 'n'. A language model can be based upon a lexicon if all the words are from a known collection. In [29], a text line image is matched against the lexicon entries to obtain a reliable segmentation and retrieve valid text. When there is no lexicon, but a corpus exists, we can build a language model based on the corpus, such as a n-gram. In [30], the authors test n-gram models on characters, words and combinations of the two.

The candidate lattice as described has some problems. First, noise in the

image may be considered a character part and cause misclassification. Second, when building the lattice the order of the segments is determined before the construction. In cases where broken characters overlap (in relation to x axis) finding the correct order is not trivial. Wrong order of parts prevents the joining of the correct segments. Our work solves such problems with an innovative approach.

2.5 Strings search

Traditional searching in a text image is usually performed on the OCR-ed text using standard text retrieval techniques. However, recognition and segmentation errors in OCR limit the accuracy of a search. Improved performance can be obtained by searching for semi-equivalent queries or to perform post-processing to OCR-ed text such as error correction.

The OCR accuracy of historical documents and manuscript is in many cases too low, so alternate approaches of keyword spotting are explored. Such approaches use features of word images (e.g. shape coding [31]) to detect the query or other global operators. These approaches are too slow for on-line search in large collections.

An intermediate approach is to search in a hypotheses text of OCR as the candidate lattice. [32] retrieves documents using multiple candidate of characters and when there are uncertain characters a shape-feature is also computed. The shape-feature describes the outline of the character shape. [33] proposes a lattice-based method for keyword spotting in on-line Chinese handwriting. This method stores the N-best strings recognition results in a database and then a query is searched in the N-best strings. [34] proposes a method for keyword spotting in off-line Chinese handwritten documents using a lattice generated by over-segmentation. In word retrieval, the query word is matched with sequences of candidate patterns in the lattice and a score is determined by word similarity.

[33] proposes a fast method for keyword spotting. However, the method does not utilize all the information in the lattice. On the other hand, [34] does utilize all the information in the lattice, but this results in a slower method. In this thesis we propose a fast method that search in all the candidate letters.

2.6 The Hebrew language

Hebrew is written from right to left using 27 alphabet letters with two main scripts: square and cursive. The square script has been in use for over two thousand years without significant change in the form while the cursive script evolved constantly. Also, cursive letters tend to be circular and flexible in form and sometimes are markedly different from their square equivalents. Our work focuses on the square script.

In the square script most letters are written by two connected strokes (e.g. צ), or by one stroke (e.g. ו), besides the letters ה and ק that are formed by disconnected strokes. However, in practice there are more letters that are disconnected (e.g. נ, ש and in Figures 1,17). Another property is that some letters are contained in other letters (e.g. י ⊂ ר ⊂ כ, ה ⊂ ם). An additional property is that there are no vowels (e.g. in English: A,E,I,O,U), and instead there are diacritical signs (called Niquq, e.g. וְבִיתָךְ שְׁלוֹם), which are written above, below, or inside letters.

Several papers were published about Hebrew document analysis in recent years but only [35] deals with letter recognition. [35] presents a method to search for Hebrew letters in historical documents based on a dynamic time warping algorithm. The features are based on profiles of small windows and the slope of a contour.

Additional work was published over 20 years ago. These works made under many technical and computational limitations and therefore their abilities are limited.

[36] presents a system that helps analyze Hebrew square script in manuscripts and includes a letter recognition mechanism based on a series of writing rules that were manually adjusted. The presented system is unfinished and requires human involvement in several stages.

[37] presents an algorithm in the syntactic approach for recognition of cursive one-stroke Hebrew letters. That is, letters are represented by structures such as lines, arcs and loops. The algorithm relies on the topological structure of the specific script and cannot be adapted to styles with unclear structure.

[38] deals with recognition of hand-printed Hebrew characters. The recognition has several stages based on skeleton, end-points, structural analysis and features extracted in the Hough transform space. This work also relies on a

specific script and is highly dependent on the skeleton that is very sensitive to noise.

Commercial OCR software for Hebrew exists from 90s and is commonly used. In recent years there is open-source OCR software. These are appropriate only for printed documents and do not work on degraded manuscripts.

In this thesis we present an OCR system designed for historical Hebrew manuscripts.

3 Document preprocessing

The preprocessing stage is to prepare the manuscripts' images for the OCR system. The difficulties in this stage stem from the fact that the manuscripts found in the Cairo Genizah demonstrate a wide variety of script types, letter sizes, materials (vellum, paper...), etc. Moreover, the manuscripts are found in various degrees of degradation and quality. These variations are seen in the manuscripts' images, which, in addition, are taken from different angles. Our preprocessing consists of several steps which are shown in Figure 2 and are described in this section.

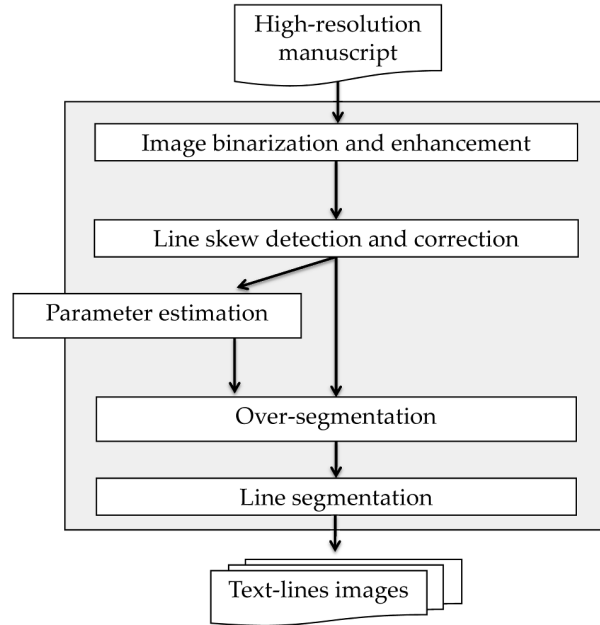


Figure 2: Document preprocessing steps

3.1 General preprocessing

The first step is to convert a manuscript image to a binary image. To enhance the binarization, morphological operations are performed.

The second step is to horizontally align the image. This is done by finding the angle of the text lines in the image. Finding this angle is done by analysing the Hough Transform of the image, and then rotating it so that the text lines

are horizontal. This step can also be repeated at the end of the preprocessing for each line separately to ensure that all the lines are be horizontal.

The next step is to compute parameters such as average line height, characteristic letter width, and stroke thickness. These parameters are used by the OCR system.

The next step of the preprocessing is to over-segment the image in order to deal with touching letters. This step is explained in detail in the next section. Finally, the lines are extracted and sent separately to the OCR system, which handles one line at a time.

3.2 Creating over-segmentation

In the over-segmentation step, the input is the connected components (CCs) of the binary image. A potential problem that arises in recognizing a text line, is that touching letters may be recognized as a single segment, thus, we need to separate touching letters. Our solution is to break the CCs of touching letters, so that each segment belongs to a single letter. However, a segmentation with too many small segments will be unusable due to computational difficulties.

The first step towards creating an over-segmentation is to detect the CCs that contain more than one letter. A possible approach is to use a letter classifier to recognize “suspect” CCs, relying on the intuition that a letter classifier will not successfully classify CCs that contains more than one letter. Thus, CCs that were not successfully classified are reasonable suspects and require over-segmentation. Another approach is to select the large CCs as suspects. As we are dealing with Hebrew manuscripts, we preferred this approach. This is because in Hebrew, there are several pairs of letters that when touching, are very similar to a single letter (e.g. $\text{ת} \sim \text{ת}$, as in Figure 3a).

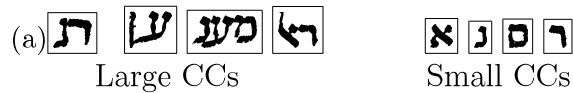


Figure 3: Examples of large and small connected components. (a) is example of a letter that might be two touching letters.

The next step, after selecting suspect CCs, is to break them into parts. This is done by a procedure that takes a binary or gray-scale image of each suspect CC,

and builds an *energy-map* for it. In an energy-map, the background is ‘0’, and the foreground pixels value are determined by the pixels’ intensity. The energy-map is for the bounding box of the CC image.

The procedure splits the CC with top-down cuts. A *cut* is an 8-connected path of pixels, running from top to bottom, containing exactly one pixel in each row (see Figure 4). Usually, the letters that compose a suspect CC are from the same line, so these cuts are appropriate. In case of a CC that crosses between lines a cut needs to be made from the left of the image to the right. In these cases we use the transposed image. The aim of the cuts is to over-segment a CC and thus to split the touching letters.



Figure 4: Example of a vertical cut.

The minimum cut, which minimizes the weight of the path on the energy-map, splits touching letters that have a small touching region with each other, i.e when the thickness of the touching region is less than the stroke’s thickness. The procedure finds the minimum cut, then increases the energy around the path, in order to prevent future cuts from being too close to the current one. The procedure repeats finding additional minimal cuts in the updated energy-map. Finally, cuts that create very small segments are rejected. An example of this process is shown in Figure 5.

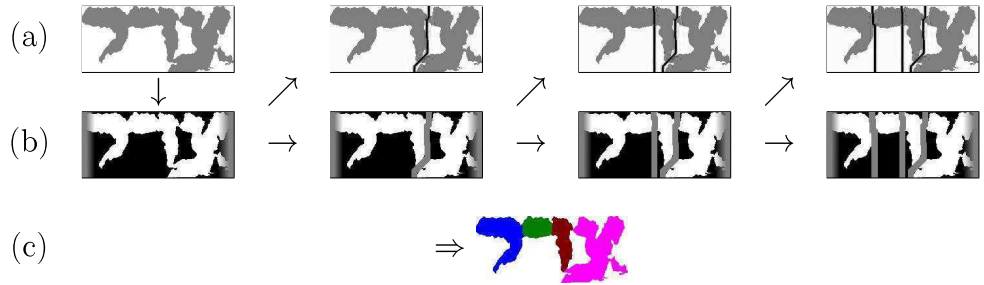


Figure 5: Demonstration of the over-segmentation process on a CC. (a) The CC image (in gray) with the cuts (in black). (b) The energy-map at each iteration. (c) The result: each segment a different color

The horizontal margins of the energy-map tend to have low energy. Therefore, we add weights to the sides of the energy map (see Figure 5b) so that the cuts are not made close to the horizontal margins of the CC.

The proposed procedure is suitable for “easy” cases of touching letters, that is, letters with a small touching region. We used heuristics based on distance transform of a binary image for the initialization of the energy-map in order to encourage the cuts to pass through noise. Other common cases of touching letters, such as continuous strokes that are typical of cursive style, can be handled by initializing the energy-map using various heuristics. Such heuristics can be based on junctions in the skeleton or other skeleton and contour analysis.

4 Reading-graph model

The reading graph is based on an over-segmented text line, as explained in Section 3.2. We compute candidate letters from the segments. Every candidate is composed from a set of proximate segments (a set may be a singleton). Every candidate is sent to a letter classifier, which returns the probability distribution of the labels for this candidate letter. We define a *join* to be a set of segments with a high probability label. Joins are stored in a *reading graph*, as depicted in Figure 6. The reading graph captures several alternative readings of the line and the plausible readings are computed from this graph.

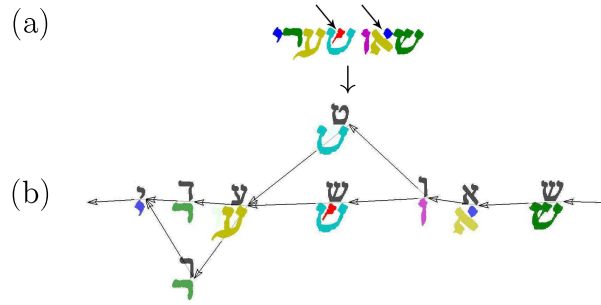


Figure 6: (a) The segments are in different colors: In this simple example most letters are composed of one segment; (b) The reading graph: The vertices are the joins (CCs of different colors) with their Hebrew letter label (black).

The reading graph is a novel data structure with similarities to the candidate lattice [28]. However, the joins of a reading graph do not necessarily include all the segments. Every vertex in the graph represents a join, proximate segment set, with a high probability letter label. Every edge in the graph connects a pair of neighboring letters. Several joins can share the same segment (e.g. Figure 7b). In addition, a segment can be part of more than one join with different labels (e.g. Figure 7a).

Accordingly, every path in the reading graph is a possible reading of the text line. We now proceed to describe the construction of the reading graph and how to extract a reading from it.

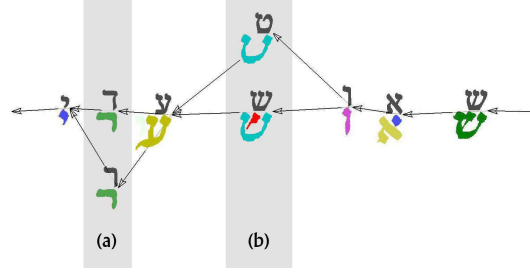


Figure 7: (a) Ambiguous shapes; (b) Joins can share same segment.

4.1 Reading-graph construction

The reading graph construction is based on join recognition in an over-segmented text line. Join recognition is performed around a given position within a bounding box, as demonstrated in Figure 8 and explained later.

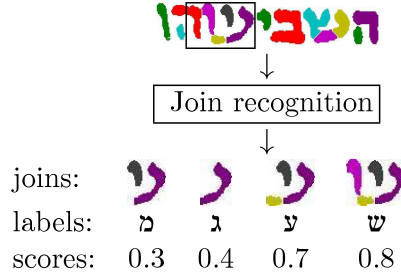


Figure 8: Join recognition in a window.

The construction starts from the beginning of the line (in Hebrew - from the right). We find possible joins for the first letter and store the joins in a reading graph. For every stored join, we continue the construction separately by recognizing the possible joins for any following letter using the remaining segments. Figure 9 demonstrates the construction process. In this figure, the two first letters have one join for each: and . While the third letter has four joins. For each of the four joins, the construction process continues separately.

Intuitively, the algorithm scans the line from right to left. At each step, the algorithm examines the current bounding box and recognizes possible joins in this box. Every join is then added as a vertex to the graph. The algorithm is called again on the rest of the line (left of the join). Edges are added to the graph such that each vertex is connected to all the vertices that correspond to joins that were recognized immediately to its left. Adding edges is done during

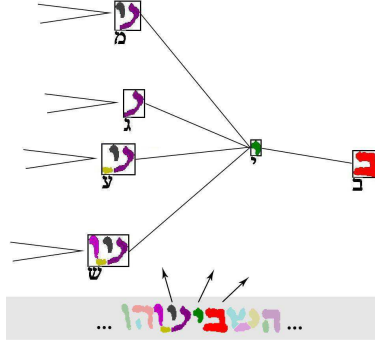


Figure 9: Example of an intermediate step of the reading-graph construction.

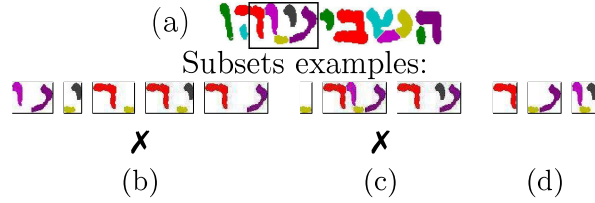


Figure 10: (a) Over-segmented text line with window for join recognition. (b) Rejected - segments are far from each other. (c) Rejected - size or location. (d) Passed the first step.

the algorithm's backtracking.

An example of a simple reading graph was shown previously in Figure 6, a more complex example is shown later in Figure 14.

4.1.1 Join recognition

Join recognition in an over-segmented text line is done within a bounding box and starting at a position (see Figure 10a). The window's dimensions are determined by a large letter based on the manuscript parameters. In order to find joins we test sets of segments which are within or partly within the window.

Testing the subsets in the search is done in two steps due to the large number of possible subsets. First, a fast and coarse operator tests if a subset may be a letter. In our implementation, the fast operator first rejects every subset in which the segments are very far from each other (see examples in Figure 10b). Then, subsets with irregular dimensions or location are rejected (Figure 10c).

The subsets which passed the fast operator are sent to a letter classifier. The classifier returns probabilities of labels (Hebrew letters). Every subset and label with a relatively high probability is a potential join. In case of ambiguous shapes (e.g. ך , ך), different potential joins can be the same subset but with different labels.

Feature extraction

The letter-classification procedure is based on features extraction. We experimented with several features based on projections, profile, chain-code, SIFT, and more. We also developed a variant of the recursive-subdivisions feature introduced in [13]. Our recursive-subdivisions feature performed significantly better than all the other features we tried, so we chose to use only this feature.

The method takes a binary or gray-scale image of the potential join and represents it with a fixed length vector. The vector holds the coordinates of recursive subdivisions which are computed by the following function. The function gets an image and computes the x-coordinate of the center of mass (first order moment). Then, the function subdivides the image into two images by the x-coordinate and calls itself (recursively) on the transpose of each of the two sub-images. The coordinate values are calculated relative to the complete image (bounding box). These coordinates are the output vector which is of length $2^d - 1$ for a recursive depth d .

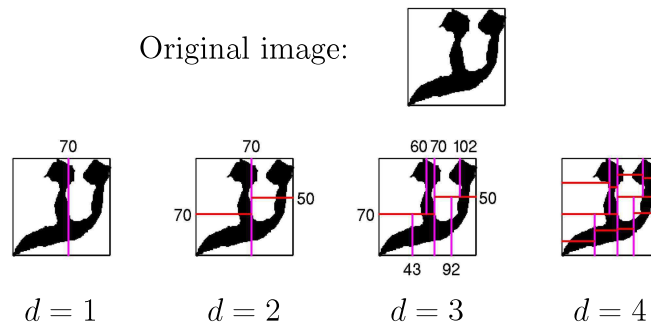


Figure 11: Demonstration of the subdivision of an image in size 122x123 .

Observe that the vector holds twice as many subdivisions along one axis than the other (indeed, for every subdivision along the x axis, we divide twice along the y axis). As a result, the representation in one direction is more granular than

the other. In order to balance the representation the function is also computed on the transposed image and the two resulting vectors are concatenated.

In order for the feature to be invariant to the size of the image it is common to resize the image to a fixed size. Resizing reduces image quality and therefore may impair the representation. An advantage of this method is the ability to normalize the vector rather than resize the original image.

In our implementation, the images consists of potential joins and the recursive depth is 8. We normalize the vector with the typical letter height of the manuscript as opposed to the common method that uses image height. Our normalization allows us to distinguish between letters that are very similar up to their height. This is important in Hebrew, due to letters such as e.g. ך, ן or ך, ן (see Figure 12).



Figure 12: The height and the width of the letters distinguish between them.

Classification

For the classification of the feature vectors of potential join, we used multi-class SVM (One-against-One LIBSVM [39]), with a radial basis function as the kernel. A grid search was performed in order to obtain the optimal values for the variance parameter of the RBF kernel and for the cost parameter of SVM using cross-validation on a training set of letters. Most of the example letters in the training set were extracted from historical manuscripts. The other example letters are from digital fonts ¹ whose design was influenced by Hebrew historical documents. The images were represented using the recursive subdivision described in a previous section.

4.1.2 Updating the reading graph

After construction of the reading graph, weights are added to the graph in order to evaluate reading possibilities. Recall that the vertices are joins (corresponding

¹Such as: Narkisim, Hadassah, David, Koren, FrankRuehl, and more. For more information see [40].

to letters) and that the edges represent neighboring letters. The joins are chosen based on their high probability, so we use the join probability as the vertex probability.

Edge weights are initialized according to probabilities bi-gram (pairs of letters probability) that were calculated from old Jewish texts². In addition, some edges weights are reduced, to handle segments that are suspected of being noisy, as explained in the following paragraph.

Non-letter segments are common in historical manuscripts, especially in Hebrew scripts, due to Niqud (see Section 2.6). The preprocessing step attempts to clean the noise, but it is not always possible to clean it all. Furthermore, small segments can be a result of over-segmentation, so a small segment may either be noise or part of a letter. A join-recognition mechanism is supposed to handle the problem of small segments by choosing joins that are similar to letters. Unfortunately, when a join of segments is contained in another join (e.g. Figure 7b: $\cup \subset \cup$), we insert to the reading graph joins that are not necessarily correct, but have a high probability.

A possible solution is to penalize a join that is contained in another join (e.g. \cup). The problem with this solution is that we might penalize a join for not using a segment, while the unused segment is actually part of the following join. Thus, a penalty should depend on whether there are unused segments in a *pair* of consecutive joins. Accordingly, we penalize an edge and not a vertex. Figure 13 demonstrates the need for a penalty that depends on the pair of consecutive joins. The blue dashed edges are penalized because of the unused segment.

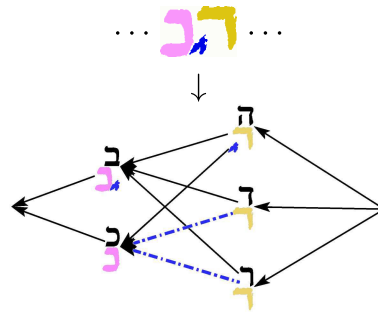


Figure 13: The reading graph of the above text. The two blue dashed edges were penalized because of the unused segment.

²Bible, Mishna and Talmud

4.2 Reading-graph applications

The reading graph can be used in several useful applications. In this section we describe three applications. (1) Finding the optimal reading in order to return text. (2) Interactive checking and correcting of output text. (3) Efficient storage and fast retrieval.

4.2.1 Finding the optimal reading

In this section we describe how to calculate the most plausible reading from a reading graph. As explained earlier, a path in the reading graph from the root (the first letter) to a leaf (the last letter) corresponds to a sequence of letters. An example of a path is shown as a blue line in Figure 14. In this graph, the vertices' weights describe the accuracy of the labels, and the edges' weights describe the plausibility that two neighboring letters are consecutive.

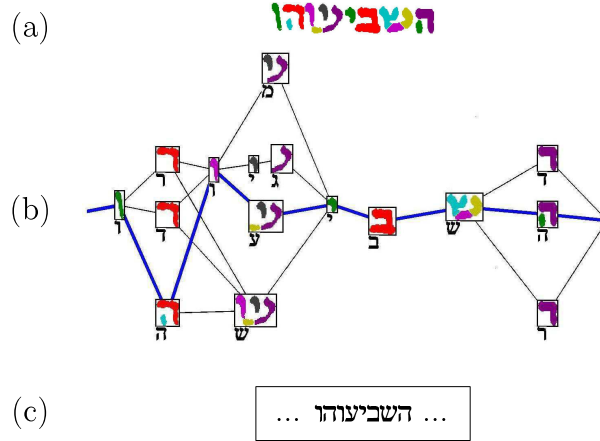


Figure 14: The reading graph of (a). The blue line is the optimal path. (c) the reading result.

Intuitively, the most plausible reading is the path that maximizes the sum of the weights along it. This, however, is generally incorrect as it creates a bias toward longer paths. That is, preferring to split letters as much as possible. A correction for this is to normalize the total weight by the length of the path. Thus, we search for a path that maximizes the average weight along it. This is formulated as the following optimization problem.

$$path^* = \arg \max_{P \in \text{Paths}} \left\{ \frac{\sum_{e \in P} weight(e)}{|P|} \right\} \quad (1)$$

To solve this optimization problem, we observe that the expression can be equivalently formulated as follows.

$$\max_{P \in \text{Paths}} \left\{ \frac{\sum_{e \in P} \text{weight}(e)}{|P|} \right\} = \quad (2)$$

$$\max_k \left\{ \max_{P \in \text{Paths of length } k} \left\{ \frac{\sum_{e \in P} \text{weight}(e)}{k} \right\} \right\} \quad (3)$$

A dynamic programming algorithm can solve this optimization problem in $O(|E|^2)$, where $|E|$ is the number of edges.

4.2.2 Interactive checking and correcting of output text

In some cases, we can trust OCR systems to output text that is “close enough” to documents, and such results are good enough. In many cases, however, an accurate transcription is very important. An accurate transcription can be obtained manually, but this may take a long time. Another option is to manually correct OCR-ed text with respect to the original manuscript. However, this also takes a long time. Other methods take problematic words and ask humans to recognize them (e.g over the internet in the form of a game or reCAPTCHA [41]) thus filling the gaps in the OCR-ed text.

Most of the Cairo-Genizah manuscripts can be read correctly only in context and only by few people. We developed an interactive tool to obtain accurate transcriptions that is aimed at the Genizah manuscripts. The tool uses the result of the optimal path as an initial transcription and allows an expert to easily change the path in the reading graph, interactively. The goal is to direct the expert and free him from a tedious search of errors. Figure 15 presents a screenshot of the tool.

The basic idea is to show an expert the text-line image and the transcription obtained from the optimal path of the reading graph. The transcription is revealed step by step, depending on the expert’s action. When there is a piece of text with only one path in the reading graph, we place all letters of the piece exactly below its sources. If there is a mistake the expert can fix it with a very simple interface. In other cases, where the text is ambiguous, the reading

graph captures several alternative readings. Then, we want to validate carefully the correctness of the letters along. Therefore, when there is a vertex in a reading graph with several sons with high scores, we present the alternatives and let the expert choose between them with a very simple interface. When an alternative is chosen, the rest of the path is computed again with respect to the new selection



Figure 15: Screen-shot of the interactive tool

The interactive tool for checking and correction of a transcription of a manuscript was tested by us. We tried either the manual way (typing the text manually) or manually corrected OCR-ed output text with respect to the original manuscript. When the OCR-ed text contains few errors, it was very hard to find the errors and to correct them and we found that our interactive tool is better. In case there are a lot of errors, it is not clear which is better, and it depends upon the user.

4.2.3 Efficient storage and fast retrieval

In this section we present a method for fast retrieval of text line of documents, according to a user query, from a large collection of documents. The documents are represented off-line as reading graphs and the reading graphs are stored in an index repository as described below. The search for a user query is performed in the index repository, which contains information about alternative readings. This way, results can be found in one of the alternative readings, even in a degraded manuscript.

Reading-graphs storage

Searching for a user query in the reading graphs does not necessarily require all the data contained in the graphs. The relevant details are the edges (neighboring-letter pairs) and their position in a text line. In addition, the edges weights are also useful for ranking the results.

An *inverted list* is an index data structure that holds a mapping from words to locations in the text. We construct an array of inverted lists for all bi-grams

(pairs of letters: e.g. aa, ab, ac, ... zz) in the language (in Hebrew there are 27^2 bi-grams), as presented in Figure 16. The index repository holds the relevant details of the reading graphs. Recall that a vertex in the reading graph corresponds to a letter and an edge connects two vertices and thus represents two consecutive letters (a bi-gram). For every edge in every reading graph we store the following information in the corresponding inverted list: the ID of the text line in which the edge appears, the position in the line, and the score of the edge. The data is stored sorted by the ID of the text line and secondly by the position in the text line.

א א	→	3, 6, 7, 22, 34, 97, ...
א ב	→	2, 12, 17, 32, 34, ...
א ג	→	1, 9, 9, 55, 34, 87, ...
⋮		
ת ת	→	6, 7, 16, 38, 74, 127, ...

Figure 16: An array of inverted lists.

Space complexity analysis: The size of all the inverted lists is large and depends on the number of documents and on the ambiguities in segmentation and recognition in the documents. On average there are 15 text lines per document in our documents and there are on average about 50 edges in a reading graph. For each edge we store the ID, position and score. In our naive implementation, storing this data for each edge takes 10 bytes. Therefore, the total memory used for storing one document is: $15 \cdot 50 \cdot 10 = 7,500$ Bytes. storing a collection of 10,000 documents takes about 75 Megabytes. This can be easily kept in main memory for quick access.

The retrieval

The retrieval of a user query is based on the inverted lists. Given a query string, we decompose it to bi-grams and intersect the inverted lists of all the bi-grams and get the text lines that contain the query. The proposed method is fast and compatible with real-time retrieval in large collections.

Intersecting the lists is done with respect to the IDs and the positions, such that each two consecutive bi-grams of the same ID need to have an overlapping

letter in the text line. This way, we force all the bi-grams to be consecutive, as are the bi-grams in the user query.

The intersection algorithm is applied to pairs of inverted lists that have an overlapping letter. For efficiency reasons, the order of the lists that we intersect is determined by taking the shortest list each time. The intersection of two inverted lists is done using two pointers to the elements of the two lists. At every step, the pointer that points to the element with the lower ID is incremented. When the two pointers point to the same ID, the positions of the elements are examined and if there is an overlap between the two bi-grams in the text line, the element is added to the intersection. The time complexity of taking the intersection is $O(n)$, where n is the sum of the lengths of the inverted lists that we intersect.

A wildcard character ('*') can be used to represent any sequence of letters, and is useful for querying. To allow wildcards in the middle of a string, we perform two queries: the prefix (the string preceding the wildcard) and the suffix (the string following the wildcard). Then, we intersect the results in order to find the cases where the prefix appears before the suffix.

Improvements

The results of a query are ranked by average edge score taken over the edges that are stored together with the same ID in the inverted lists. This rank is the score of the string's path in the reading graph and describes the quality of the string that is found. In rare cases, the query string does not exist exactly in the text line. Such cases may occur when there are two overlapping bi-grams whose corresponding edges in the reading graph do not share a vertex. Since we do not store the vertices of the reading-graphs we do not detect such cases. However, as we mentioned, these cases are rare (at least in our manuscripts). It is possible to load the reading graph and check that the returned string induces a connected path in the graph, however, this slows down the query search.

Finally, the results are presented to the user in decreasing order of rank. However, hundreds of possible results for a query are too much and usually a user is interested only in the few top results. A possible improvement is to retrieve only the top results and retrieve additional results upon request from the user. Implementing this can be done by dividing inverted lists to chunks according to score ranges. Then, we can start the search with the highest scoring chunks and continue upon request from the user to the other chunks.

Sorted lists can be stored in a B+ tree instead a simple list. A B+ tree represents sorted data in a way that allows efficient intersection with a small memory overhead for pointers. Another basic improvement is to use in addition inverted lists which correspond to popular words. In such inverted lists we can store the result of popular searches and avoid computing of them every time.

5 Experimental results

We evaluated the performance of the proposed methods in several ways. The feature extraction was tested on a character database as explained in section 5.1. The accuracy (in edit distance) of the optimal path in the reading graph was tested on historical manuscripts that were written in square style as explained in section 5.2. The string retrieval was tested on the same manuscripts as the reading graph as well as on synthetic documents as described in section 5.3.

5.1 Feature extraction

The evaluation of the recursive-subdivisions feature was done by measuring the classification error of a multi-class SVM. For our experiments we used two character databases: MNIST Database, and a Hebrew-letters database we collected.

The MNIST Database consists of 70,000 isolated, handwritten digits. All the digit images are normalized to a 28x28 matrix and are represented by the recursive-subdivisions feature. The MNIST database is divided into a training set of 60,000 and a test set of 10,000 digits. We trained an SVM classifier with an RBF kernel on the training set and we got on an error of 1.57% on the test set. It is worth mentioning that in our system, we take into account not only the highest ranking label, but the second and third as well (when constructing the reading graph). In these case, the error decreases to 0.4% and 0.05% respectively.

The Hebrew-letters database consists of Hebrew fonts and letters extracted manually from historical manuscripts. In order to use the feature extraction optimally we examined several recursive depths. In addition, since there are several similar Hebrew letters up to scale (see Figure 12), we examined different normalization methods.

The original method of the feature extraction includes non-uniform representation as explained earlier in Section 4.1.1. In order to overcome the non-uniform representation we proposed to repeat the computation with the transposed image. We examined the utility by comparison of vectors with the same length.

The normalization methods we examined are: (1) Normalizing to a fixed-size square; (2) Normalizing to a fixed-size rectangle by image height (keep the aspect ratio); (3) The same as 2, but using typical height of the letters; (4) Concatenation

of 2 and 3 from the previous depth.

Table 1 summarizes the performance of the feature extraction in the different settings. The normalization method using typical height achieved a lower error and the method that combined the two basic methods was the best. Most of the comparisons showed improvement in the classification error when we also used a transposed image.

Normalization method:		1x1		Letter hight		Line height		Combination	
Including transpose?		X	✓	X	✓	X	✓	X	✓
Vector length:	2^5	3.07	3.34	2.92	2.96	2.24	2.46	2.73	4.47
	2^6	2.88	2.62	3.00	2.50	2.09	1.82	2.01	2.01
	2^7	2.77	2.69	2.58	2.28	1.90	1.97	2.05	1.67
	2^8	3.07	2.65	2.96	2.62	2.16	1.86	1.97	1.71

Table 1: The performance of feature extraction on a Hebrew-letters database. We tested four nomalization methods, on varying vector lengths. Each test was done with and without using the transposed image.

5.2 Handwriting recognition

We tried the system on various degraded manuscripts: 104 pages of a hand-written book and an additional 202 different manuscripts. Figure 17 presents examples of word images from the manuscripts. All the manuscripts were written in square style and are from the Cario Genizah (www.genizah.org).

The manuscripts of the hand-written book had of many diacritical signs (e.g. וְכִיִּתָּהּ שְׁלוֹם), degraded letters and noise. The other manuscripts have the same problems and in addition they were written by many writers and thus include a wide variety of the square style and letter sizes. Table 2 shows the distribution of the letters in several manuscripts we tested.

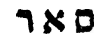





	examples	page 1	page 2	page 3
complete letters		66%	69%	37%
legitimate disconnected letters (e.g. j)		4%	7%	6%
disconnected letters		3%	8%	20%
touching letters		9%	8%	24%
degraded letters		12%	7%	6%
unrecognized		6%	1%	7%
Total normal letters		70%	76%	43%
Total abnormal letters		30%	24%	57%

Table 2: Distribution of letter types in a degraded manuscripts.

The overall recognition of our OCR system is calculated using the *edit distance* (also known as Levenshtein distance). The edit distance between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. We used it to measure the similarity between the OCR-ed text produced by our method and the ground truth text. The overall recognition rate is defined by equation 4. Table 3 presents the recognition rates for the three pages that are mentioned above.

$$\text{Recognition rate} = 1 - \frac{\text{Edit distance}}{\text{Number of letters in page}} \quad (4)$$

	page 1	page 2	page 3
Working time (sec)	93	81	181
Letters number	344	359	364
Edit distance	29	21	44
Recognition rates	91.6%	94.2%	88.0%

Table 3: Recognition rate of OCR-ed text produced by the optimal path.

5.3 String retrieval

The string retrieval algorithm was tested on two collections, each in a different way. In both collections, the documents were represented by reading graphs that were stored in inverted lists, as described in Section 4.2.3.

The first collection includes all the manuscripts that were mentioned in the previous section (104 + 202 pages). Since we do not have a ground-truth for these manuscripts, we manually checked the result as follows. We arbitrarily chose strings (with 4-9 letters, see examples in Figures 17) and denoted their position. Then, we searched for the strings and checked if the chosen positions were in the results. Figure 17 presents images that we searched for their string and Table 4 presents the results.

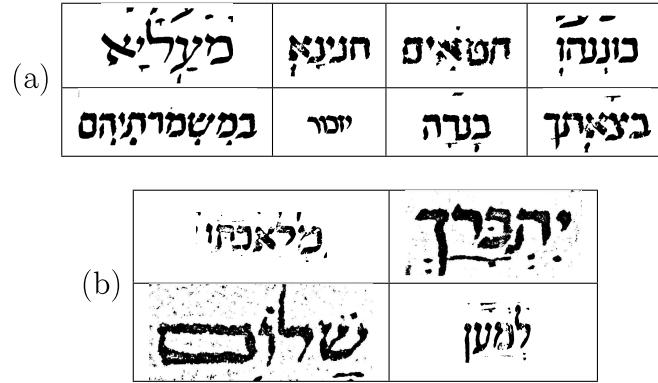


Figure 17: Word examples from the Cairo Genizah. (a) Successful examples of search. (b) Unsuccessful examples of search.

Length	# found	# total	Found rate	avr time (sec)
4	31	36	86.11%	0.08
5	26	31	83.87%	0.13
6	25	33	75.76%	0.15
7-9	23	31	74.19%	0.15
Total:	105	131	80.15%	0.13

Table 4: Retrieval results from a collection of degraded manuscripts.

The second collection consists of synthetic text images of a well-known book. In order to complicate the reading graph we degraded the image by adding noise that connects and breaks letters, such that there is a large number of alternative readings.

In order to evaluate the recall and the precision of our method, we randomly chose 89 words with 3-7 letters. Then, we performed a search for these words. The results and computation-time are presented in Table 5 and are measured by recall, precision and F-Measure, defined as follows:

$$\text{Recall} = \frac{\# \text{ correct detected words}}{\# \text{ truth words}} \quad (5)$$

$$\text{Precision} = \frac{\# \text{ correct detected words}}{\# \text{ detected words}} \quad (6)$$

$$\text{F-Measure} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (7)$$

Words length	Recall	Precision	F-Measure	Time (sec)
3	1.00	0.71	0.81	0.0166
4	0.99	0.84	0.90	0.0437
5	0.97	0.97	0.96	0.0664
6	0.96	1.00	0.98	0.0759
7	0.96	1.00	0.98	0.0825

Table 5: Retrieval result in terms of recall, precision and f-measure.

6 Summary

Throughout this thesis we presented our system for handwriting recognition and fast retrieval of Hebrew historical manuscripts. Our system efficiently finds a plausible reading of the text image, offers alternative readings (via an interactive tool), and allows searching for strings in every alternative reading of every manuscript in a large collection.

Degraded manuscripts contain many abnormal letters (degraded, disconnected, and touching), as well as variance in handwriting. We developed a method for creating over-segmentation in touching characters, which allows us to deal with such abnormalities. See Section 3.2 for details.

To decide which segments compose the letters in an over-segmentation of the image, we could use the candidate lattice (presented in Section 2.4). However, there are problems in using this structure (see section 2.4). Therefore, we propose a novel structure - the reading graph (explained in Section 4), which is more flexible as and enables us to overcome the problems we have with the candidate lattice.

The reading-graph construction is done by a recursive algorithm that recognizes joins (proximity segments sets, which are similar to letters). The join recognition is done by a method for feature extraction, which we improved upon (see Section 1). The joins that are found, and their respective scores, are stored in a reading graph together with the relations between them (see Section 4). In addition, the probabilities of bi-grams and information about unused segments are incorporated in the reading graph. Incorporating all this information, the reading graph captures alternative readings of the text image.

We developed three applications for our reading graph. (1) Finding the optimal reading in order to return plausible text. (2) Interactive checking and correcting of output text. (3) Efficient storage and fast retrieval of manuscripts. Experimental results on the Cairo Genizah show that our system successfully reads degraded manuscripts containing many abnormal letters, as well as retrieves manuscripts even when the image of the query string is degraded.

References

- [1] Mohamed Cheriet, Nawwaf Kharma, Cheng-Lin Liu, and Ching Suen. *Character Recognition Systems: A Guide for Students and Practitioners*. Wiley-Interscience, October 2007. (1)
- [2] A. K. Jain, R. P. W. Duin, and Jianchang Mao. Statistical pattern recognition: a review. *Pattern Analysis and Machine Intelligence*, 22(1):4–37, January 2000. (1)
- [3] H. Fujisawa. Forty years of research in character and document recognitionan industrial perspective. *Pattern Recognition*, 41(8):2435–2446, August 2008. (1)
- [4] R. Plamondon and S. N. Srihari. Online and off-line handwriting recognition: a comprehensive survey. *Pattern Analysis and Machine Intelligence*, 22(1):63–84, January 2000. (2)
- [5] E. Grosicki and H. El Abed. ICDAR 2009 Handwriting Recognition Competition. In *10th International Conference on Document Analysis and Recognition*, pages 1398–1402, 2009. (2)
- [6] S. Espana Boquera, S. Espana Boquera, M. J. Castro Bleda, M. J. Castro Bleda, J. Gorbe Moya, J. Gorbe Moya, F. Zamora Martinez, and F. Zamora Martinez. Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33:767–779, April 2011. (2)
- [7] K. Ntzios, B. Gatos, I. Pratikakis, T. Konidakis, and S. J. Perantonis. An old greek handwritten OCR system based on an efficient segmentation-free approach. *International Journal on Document Analysis and Recognition*, 9:179–192, April 2007. (2.1, 2.2, and 2.3)
- [8] G. Vamvakas, B. Gatos, N. Stamatopoulos, and S. J. Perantonis. A Complete Optical Character Recognition Methodology for Historical Documents. In *DAS '08. The Eighth IAPR International Workshop on Document Analysis Systems, 2008.*, pages 525–532, 2008. (2.1 and 2.3)
- [9] Markus Diem and Robert Sablatnig. Are Characters Objects? In *2010 12th International Conference on Frontiers in Handwriting Recognition*, volume 0, pages 565–570, Los Alamitos, CA, USA, 2010. IEEE Computer Society. (2.1 and 2.3)
- [10] Dayong Zhu and Xiaoguang Tian. A Novel System Model to Recognition of Merged-Characters Under Linear, Nonlinear and Overlapped. In *International Conference on Technologies and Applications of Artificial Intelligence*, volume 0, pages 31–38, Los Alamitos, CA, USA, 2010. IEEE Computer Society. (2.2 and 2.3)

- [11] G. Vamvakas, B. Gatos, S. Petridis, and N. Stamatopoulos. An Efficient Feature Extraction and Dimensionality Reduction Scheme for Isolated Greek Handwritten Character Recognition. In *9th International Conference on Document Analysis and Recognition (ICDAR'07)*, pages 1073–1077, Washington, DC, USA, 2007. IEEE Computer Society. (2.2)
- [12] U. Pal, N. Sharma, T. Wakabayashi, and F. Kimura. Handwritten Numeral Recognition of Six Popular Indian Scripts. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition*, volume 2, pages 749–753, Washington, DC, USA, 2007. IEEE Computer Society. (2.2)
- [13] A. Sexton, A. Todman, and K. Woodward. Font recognition using shape-based quad-tree and kd-tree decomposition. In *Proceedings Of The Joint Conference On Information Sciences*, volume 5, pages 212–215, 2000. (2.2 and 4.1.1)
- [14] Jaehwa Park, V. Govindaraju, and S. N. Srihari. OCR in a hierarchical feature space. *Pattern Analysis and Machine Intelligence*, 22(4):400–407, April 2000. (2.2)
- [15] Alan P. Sexton and Volker Sorge. Database-driven Mathematical Character Recognition. *Graphics Recognition. Ten Years Review and Future Perspectives*, pages 218–230, August 2006. (2.2)
- [16] Georgios Vamvakas, Basilis Gatos, and Stavros J. Perantonis. Handwritten character recognition through two-stage foreground sub-sampling. *Pattern Recognition*, 43(8):2807–2816, August 2010. (2.2)
- [17] Cheng-Lin Liu and Hiromichi Fujisawa. Classification and Learning Methods for Character Recognition: Comparison of Methods and Remaining Problems. In Simone Marinai and Hiromichi Fujisawa, editors, *Machine Learning in Document Analysis and Recognition*, volume 90 of *Studies in Computational Intelligence*, pages 139–161, Berlin, Heidelberg, 2008. Springer Berlin / Heidelberg. (2.2)
- [18] Cheng-Lin Liu, Kazuki Nakashima, Hiroshi Sako, and Hiromichi Fujisawa. Handwritten digit recognition: benchmarking of state-of-the-art techniques. *Pattern Recognition*, 36(10):2271–2285, 2003. (2.2)
- [19] Francesco Camastra. A SVM-based cursive character recognizer. *Pattern Recogn.*, 40:3721–3727, December 2007. (2.2)
- [20] Tanzila Saba, Ghazali Sulong, and Amjad Rehman. A Survey on Methods and Strategies on Touched Characters Segmentation. *International Journal of Research and Reviews in Computer Science*, 2010. (2.3)

- [21] Safwan Wshah, Zhixin Shi, and Venu Govindaraju. Segmentation of Arabic Handwriting Based on both Contour and Skeleton Segmentation. pages 793–797, July 2009. (2.3)
- [22] Xianghui Wei, Shaoping Ma, and Yijiang Jin. Segmentation of Connected Chinese Characters Based on Genetic Algorithm. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, ICDAR '05, pages 645–649, Washington, DC, USA, 2005. IEEE Computer Society. (2.3)
- [23] G. Congedo, G. Dimauro, S. Impedovo, and G. Pirlo. Segmentation of numeric strings. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, volume 2 of *ICDAR '95*, pages 1038–1041 vol.2. IEEE Computer Society, August 1995. (2.3)
- [24] Rui Ma, Yingnan Zhao, Yongquan Xia, and Yunyang Yan. A Touching Pattern-Oriented Strategy for Handwritten Digits Segmentation. In *International Conference on Computational Intelligence and Security, 2008. CIS '08*, volume 1, pages 174–179. IEEE, December 2008. (2.3)
- [25] Nija Babu, N. G. Preethi, and S. S. Shylaja. Degraded Document Image Enhancement Using Hybrid Thresholding and Mathematical Morphology. In *Graphics & Image Processing, Sixth Indian Conference on Computer Vision*, volume 0, pages 701–705, Los Alamitos, CA, USA, 2008. IEEE Computer Society. (2.3)
- [26] Donggang Yu and Hong Yan. Reconstruction of broken handwritten digits based on structural morphological features. *Pattern Recognition*, 34(2):235–254, 2001. (2.3)
- [27] V. Lavrenko, T. M. Rath, and R. Manmatha. Holistic word recognition for handwritten historical documents. In *Document Image Analysis for Libraries*, pages 278–287. IEEE, 2004. (2.3)
- [28] H. Murase. Online recognition of free-format Japanese writings. In *ICPR*, pages 1143–1147 vol.2. IEEE, November 1988. (2.4 and 4)
- [29] Cheng L. Liu, Masashi Koga, and Hiromichi Fujisawa. Lexicon-Driven Segmentation and Recognition of Handwritten Character Strings for Japanese Address Reading. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:1425–1437, 2002. (2.4)
- [30] Qiu-Feng Wang, Fei Yin, and Cheng L. Liu. Integrating Language Model in Handwritten Chinese Text Recognition. In *2009 10th International Conference on Document Analysis and Recognition*, volume 0, pages 1036–1040, Los Alamitos, CA, USA, 2009. IEEE Computer Society. (2.4)

- [31] Shijian Lu, Linlin Li, and Chew L. Tan. Document Image Retrieval through Word Shape Coding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1913–1918, 2008. (2.5)
- [32] T. Kameshiro, T. Hirano, Y. Okada, and F. Yoda. A Document Image Retrieval Method Tolerating Recognition and Segmentation Errors of OCR Using Shape-Feature and Multiple Candidates. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition, ICDAR '99*, Washington, DC, USA, 1999. IEEE Computer Society. (2.5)
- [33] Heng Zhang and Cheng-Lin Liu. A Lattice-Based Method for Keyword Spotting in Online Chinese Handwriting. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1064–1068, 2011. (2.5)
- [34] Liang Huang, Fei Yin, Qing-Hu Chen, and Cheng-Lin Liu. Keyword Spotting in Offline Chinese Handwritten Documents Using a Statistical Model. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 78–82, 2011. (2.5)
- [35] I. Rabaev, O. Biller, J. El-Sana, K. Kedem, and I. Dinstein. Case Study in Hebrew Character Searching. In *International Conference on Document Analysis and Recognition*, 2011. (2.6)
- [36] Laurence Likforman-Sulem, Henri Maître, and Colette Sirat. An expert vision system for analysis of Hebrew characters and authentication of manuscripts. *Pattern Recognition*, 24(2):121–137, 1991. (2.6)
- [37] Lev and. Recognition of handwritten Hebrew one-stroke letters by learning syntactic representations of symbols. *IEEE Transactions on Systems, Man, and Cybernetics*, 19:1306–1312, September 1989. (2.6)
- [38] M. Kushnir, K. Abe, and K. Matsumoto. Recognition of handprinted Hebrew characters using features selected in the Hough transform space. *Pattern Recognition*, 18(2):103–114, 1985. (2.6)
- [39] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2, 2011. (4.1.1)
- [40] Ada Yardeni. *The book of Hebrew script: history, palaeography, script styles, calligraphy & design*. Oak Knoll Pr, 2002. (1)
- [41] Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science*, 321(5895):1465–1468, September 2008. (4.2.2)

A Algorithm for Reading-graph construction

This appendix presents a recursive algorithm for the construction of a reading graph that presented in Section 4.

Algorithm 1 Reading graph construction

(join is a proximity segments set, which is similar to a letter label)

```
1: function findNextJoins (curJoin, graph G)
2:   if curJoin is last-letter of the line then
3:     return
4:   end if
5:   joins = recognized joins following curJoin // Section 4.1.1
6:   for all j ∈ joins do
7:     add vertex j to G
8:     nextJoins = findNextJoins (j, G) // Recursive call
9:     add edges (j, nextJoins) to G
10:  end for
11:  return joins
12: end
```

The reading-graph is constructed using a function that recursively traverses an over-segmented text line. This function is presented in Algorithm 1. The function has two inputs - a join *curJoin*, and a reading graph *G*. The input variable *curJoin* is a join added to the graph. Initially, *curJoin* is set to *null*, which represents the beginning of the line and *G* is an empty graph.

The function recognizes the joins following *curJoin* (line 5) as was shown in Figure 8 and as we explain shortly. As a result, *joins* contains all the possibilities for the consequent joins after *curJoin*.

For every possible join *j*, line 7 adds a vertex, representing the join, to the graph *G*. Line 8 recursively calls the function with a new current join *j* and the updated *G*, and gets all the next joins. Line 9 adds edges from the vertex of join *j* to all the consequent vertices. After the loop in Line 6 finishes, the function return *joins* (line 11), which contains all the possibilities for the consequent joins to be set in *curJoin*. The joins that are in *joins* are set to *nextJoins* (line 8), for a previous recursive call. In case there are no consequent joins (line 2), the current call stops. Finally, the construction finishes after all the recursive calls are completed, and then *G* contains the reading graph.

In conclusion, the algorithm does three operations for every new join j . (1) It recognizes the joins following j . (2) It adds all the recognized joins to the graph. (3) It adds all the edges from any vertex j to the new vertices of the recognized joins. Figure 18 demonstrates it.

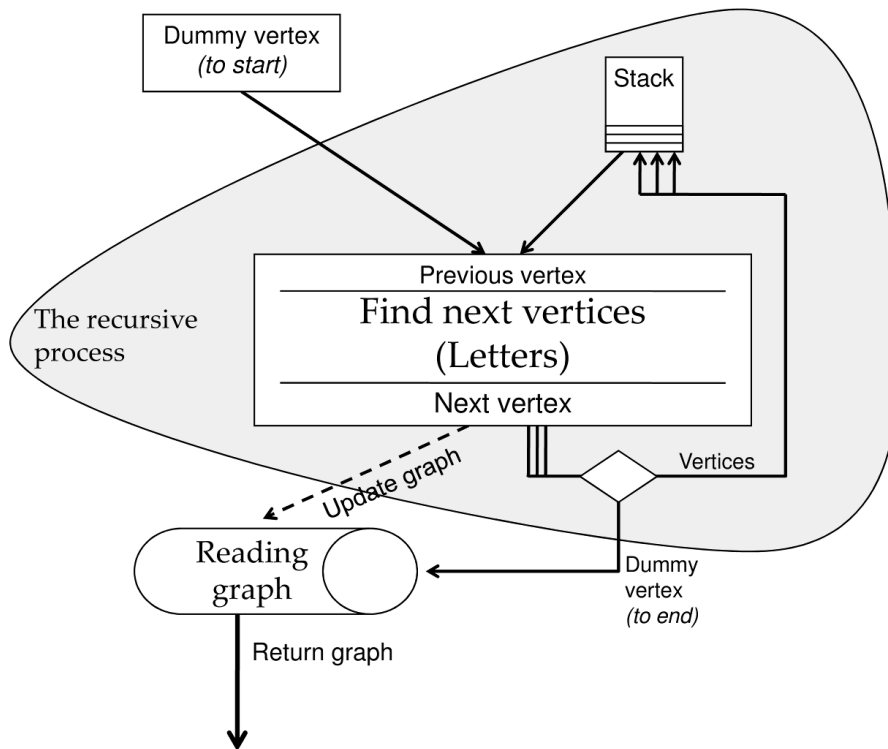


Figure 18: Flowchart of the reading-graph construction.

B Compute path with maximum average

This appendix presents a dynamic programming algorithm for computing the average maximum path in a directed acyclic graph (as the reading graph). This is formulated as the following optimization problem.

$$path^* = \arg \max_{P \in \text{Paths}} \left\{ \frac{\sum_{e \in P} weight(e)}{|P|} \right\}$$

Algorithm 2 Compute path with maximum average weight

Initialize:

$d_0(s) = 0$

$\forall v \in V \setminus s : d_0(v) = -\infty$

for $i = 2 \rightarrow n$ **do**

for all $v = 1 \rightarrow n$ **do**

$d_i(v) = \max_{u:(u,v) \in E} \{d_{i-1}(u) + w((u,v))\}$

$f_i(v) = \arg \max_{u:(u,v) \in E} \{d_{i-1}(u) + w((u,v))\}$

end for

end for

$t = \arg \max_k \left\{ \frac{d_k(last_vertex)}{k} \right\}$

$path(t) = last_vertex$

for $i = t \rightarrow 2$ **do**

$path(i-1) = f_i(path(i))$

end for

return $path$
