Handwritten Text Segmentation using Average Longest Path Algorithm

Dhaval Salvi, Jun Zhou, Jarrell Waggoner, and Song Wang
Department of Computer Science and Engineering
University of South Carolina, Columbia, SC 29208, USA
{salvi,zhouj,waggonej}@email.sc.edu, songwang@cec.sc.edu

Abstract

Offline handwritten text recognition is a very challenging problem. Aside from the large variation of different handwriting styles, neighboring characters within a word are usually connected, and we may need to segment a word into individual characters for accurate character recognition. Many existing methods achieve text segmentation by evaluating the local stroke geometry and imposing constraints on the size of each resulting character, such as the character width, height and aspect ratio. These constraints are well suited for printed texts, but may not hold for handwritten texts. Other methods apply holistic approach by using a set of lexicons to guide and correct the segmentation and recognition. This approach may fail when the lexicon domain is insufficient. In this paper, we present a new global non-holistic method for handwritten text segmentation, which does not make any limiting assumptions on the character size and the number of characters in a word. Specifically,the proposed method finds the text segmentation with the maximum average likeliness for the resulting characters. For this purpose, we use a graph model that describes the possible locations for segmenting neighboring characters, and we then develop an average longest path algorithm to identify the globally optimal segmentation. We conduct experiments on real images of handwritten texts taken from the IAM handwriting database and compare the performance of the proposed method against an existing text segmentation algorithm that uses dynamic programming.

1. Introduction

Offline handwritten text recognition has a wide range of applications, such as automatic bank check processing and handwritten postal address recognition. One major challenge in handwritten text recognition is that neighboring characters within a word may be connected when written, as shown in Fig. 1. Many OCR tools are built to recognize individual characters [7, 15, 22, 8]. As a result of this, to achieve handwritten text recognition, we often need to segment a connected word (or words) into individual characters [18], which we call *handwritten text segmentation* in this paper.

Handwritten text segmentation is a very difficult problem because there is a large variation in handwriting styles. For example, people may write the same character in different ways, including in different shapes and sizes, even within the same word, as shown in Fig. 1(a). As a result, it is usually difficult to ascertain the number of characters in a handwritten text to be segmented. Furthermore, the various ways in which two neighboring characters could be connected make it very difficult to determine the *boundary* that separates neighboring characters by evaluating only local stroke geometries.

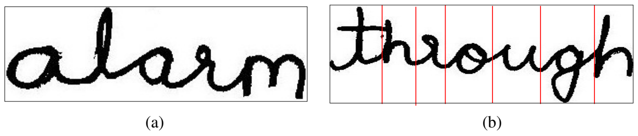

Figure 1. An illustration of the challenges in handwritten text segmentation. Note that the two "a"s show different sizes and shapes in (a) and there are different kinds of connection strokes in (b), where red vertical lines are the segmentation *boundaries* between neighboring characters.

Earlier methods for text segmentation evaluate the local stroke geometry for segmentation boundaries [12, 5]. For example, Liang *et al*. [11] propose two different types of projections to construct a segmentation, and optimize this segmentation using a dynamic recursive algorithm and contextual information. Between the top and bottom sides of the text image, Wang *et al*. [20] find paths with the minimum number of stroke pixels, and use such shortest paths as the text-segmentation boundaries. However, these overly simplified criteria for determining the segmentation boundaries work only for printed texts, but not handwritten texts.

Recent methods use character recognition for aiding text segmentation. In these methods, for each resulting text segment, a *character likeliness* is first defined to measure how likely the segment is a valid character using a character recognition algorithm. Text segmentation is then achieved when the resulting characters show large character likeliness. For example, in [16] the text image is described by a feature graph and the text segmentation is achieved by identifying subgraphs with large character likeliness. Martin *et al*. [14] use a sliding window approach to scan horizontally over the text image, and use a neural network classifier to measure the character likeliness of the subimage within the sliding window. Recently, the award winning paper (best student paper of ICDAR 2009) by Roy *et al*. [17] use local stroke geometry to identify a set of candidate segmentation boundaries, and then use dynamic programming to decide the final segmentation boundaries that lead to a maximum total character likeliness.

Many holistic methods have also been developed for text image segmentation and recognition. In these methods, it is assumed that the texts presented in the image are valid words from a set of lexicons [21, 1]. The text image segmentation and word recognition are then solved simultaneously by using features from the whole text image. For example, Arica *et al*. [3] extend the method used by [10] to segment characters by running a series of constrained shortest-path algorithms, and use a Hidden Markov Model to do word-level recognition. Lee *et al*. [9] extend the method developed in [14] by using a cascade neural network classifier. However, if the texts presented in an image are not valid words (for example, in the application of finding typos), or the lexicon domain is insufficient, the above holistic methods will fail.

In this paper, we develop a new global non-holistic method to segment handwritten text by maximizing the *average* character likeliness of the resulting text segments. Different from many previous methods, this proposed method does not make any limiting assumptions about the number of resulting characters and the size of an individual character. We uniformly and densely sample the text image to construct a set of candidate segmentation boundaries. A directed graph is then constructed to embed the character likeliness of the text segments between any two candidate segmentation boundaries. In this graph, text segmentation is reduced to the problem of finding an *average* longest path between the first and the last candidate segmentation boundary. We find that the average longest path in the constructed graph can be found in polynomial time with global optimality. We implement such an algorithm, and test it on real handwritten text images taken from the IAM handwriting database.

The remainder of this paper is organized as follows. Section 2 provides the technical details of each component of the proposed method. Section 3 reports the experimental results on real handwritten text images. A short conclusion is given in Section 4.

2. Proposed Method

In this paper, we consider binary handwritten text images with the border fitting tightly around the text. As shown in Fig. 2, the proposed method consists of several components.

1. Construct the candidate segmentation boundaries, as shown by the vertical red lines in Fig. 2(b).
2. Construct a directed graph where each vertex represents a candidate segmentation boundary, including the left and right image border, where each edge represents the text segment between two candidate segmentation boundaries, as shown in Fig. 2(c).
3. Weigh the graph edges by the character likeliness derived from a character recognition algorithm.
4. Find the average longest path between the leftmost vertex (left image border) and rightmost vertex (right image border) in the graph, as shown in Fig. 2(d).
5. Take the candidate segmentation boundaries, corresponding to the vertices along the identified average longest path, as the final segmentation boundaries for text segmentation, as shown in Fig. 2(e).

For Component 1, we uniformly and densely partition the text image, as shown in Fig. 2(c), for the candidate segmentation boundaries. When speed is not a factor, we can even partition the text image into single-column text segments. Note that the candidate segmentation boundaries constructed using this approach contain a large number of false positives compared to the set of true segmentation boundaries. While most previous methods need a good initial set of candidate segmentation boundaries (*e.g*., covering all the desired segmentation boundaries with few false positives), the proposed method can robustly handle a large number of false positives.

For Component 2, given a set of candidate segmentation boundaries $S_1, S_2, \cdots, S_n$ that are ordered from left to right on the text image, as shown in Fig. 2, we construct a directed graph $G = (V, E)$ as follows:

1. Each candidate segmentation boundary S_i will be represented by a vertex v_i in G. Note S_1 and S_n represent the left and right border of the text image.
2. Between each pair of vertices v_i and v_j, where $i < j$, we construct a *directed* edge $e_{ij} = (v_i, v_j)$ that starts from v_i and ends at v_j.

Note that we construct edges between both neighboring and non-neighboring candidate segmentation boundaries. As mentioned above, each edge represents the text segment between two candidate segmentation boundaries. Therefore, an edge between non-neighboring candidate segmentation

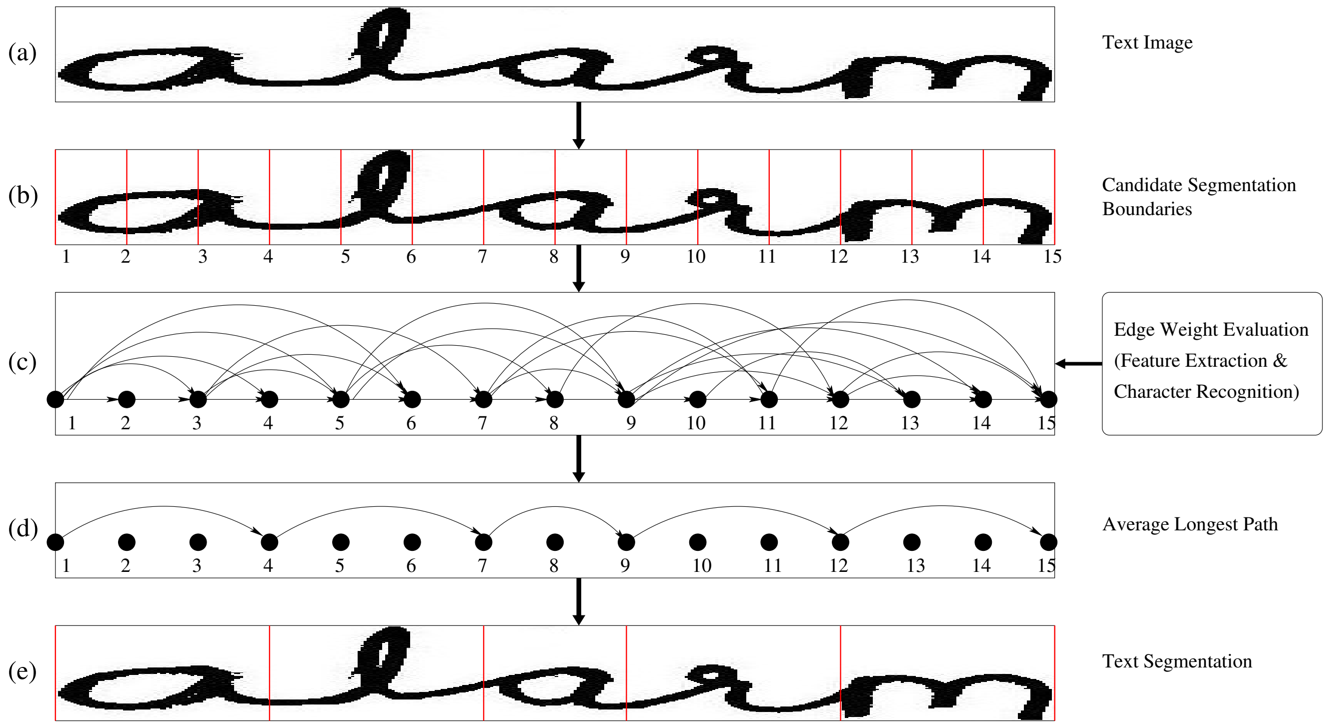

Figure 2. An illustration of the components of the proposed method.

boundaries actually represents a text segment that merges multiple partitions from Component 1. An example is illustrated in Fig. 2, where edge e_{14} actually indicates that the first three partitions from Component 1 are merged as a text segment. If this edge is included in the identified average longest path, this merged text segment will constitute a single character in the final text segmentation.

In the following, we first present a method that uses character recognition for measuring the character likeliness of a text segment, and use this character likeliness as the edge weight in the constructed graph. We then develop a graph algorithm to find the average longest path for text segmentation.

2.1. Character Likeliness Measure

The edge weight $w(e_{ij})$ describes the character likeliness of the text segment between candidate segmentation boundaries S_i and S_j, where $i < j$. The basic idea is to feed this text segment (a subimage) into an adapted character classifier to ascertain its character likeliness: a text segment fully and tightly covering a single character is expected to return a high character likeliness while a text segment covering part of a character, or overlapping multiple characters, is expected to return a low character likeliness. In this paper, we train a set of support vector machine (SVM) classifiers for this purpose.

In this paper, we focus on text consisting of the 26 Roman alphabetic characters. Thus we have 26 classes of characters. We train a binary SVM classifier [4] for each class of characters. For this purpose, we collect a set of isolated handwritten characters as training samples. In training the binary SVM classifier for a specific character class, say “a”, the training samples in this class are taken as positive samples and the training samples in the other 25 classes are taken as negative samples. When a new test sample is fed into this binary SVM classifier, we obtain a *class likeliness* associated with this character class. By testing against all 26 SVM classifiers, we obtain the class likeliness associated with each of these 26 character classes, and we simply take the maximum class likeliness as the character likeliness.

More specifically, in this paper we use the libSVM [4] implementation for each binary SVM classifier, which has two outputs: a classification indicator of positive $(+1)$/negative (-1), and a probability estimate p in $[0, 1]$ that describes the confidence of the classification. If the indicator is $+1$, we simply take p as the class likelihood. If the indicator is -1, we take $1 - p$ as the class likelihood because, in this case, p is the negative classification confidence.

For a text segment, we extract the HOG (Histogram of Oriented Gradients) based features [13] as the input for the SVM classifiers. We first normalize the size of the text segment to a 28×28 image. Each pixel in the image is assigned an orientation and magnitude based on the local gradient. The histograms are then constructed by aggregating the pixel responses within cells of various sizes. Histograms with cell size 14×14, 7×7 and 4×4 with overlap of half the cell size were used. Histograms at each level are multiplied by weights 1, 2 and 4 and concatenated together to form a single histogram. The details of these feature construction can be found in [13].

2.2. Text Segmentation by Finding Average Longest Path

Based on the above formulation, the major task of text segmentation is to identify a subset of candidate segmentation boundaries $S_{f_1}, S_{f_2}, \cdots, S_{f_m}$, where $1 = f_1 < f_2 < \cdots < f_m = n$, as the final segmentation boundaries. The number of final segmentation boundaries, m, is unknown, and to be discovered in text segmentation. The general principle is that the text segments defined by boundary pairs S_{f_i} and $S_{f_{i+1}}$, $i = 1, 2, \cdots, m$ should show large character likeliness. In another words, the graph edges between v_{f_i} and $v_{f_{i+1}}$ should have a large weight. In [17], this is formulated as an optimization problem

$$\begin{aligned}&(m^*, f_i^*; i = 1, 2, \cdots, m^*)\\ =\ &\arg \max_{m, f_i; i=1,2,\cdots,m} \sum_{i=1}^{m-1} w_{f_i, f_{i+1}},\end{aligned}$$

where $w_{f_i,f_{i+1}} = w(v_{f_i}, v_{f_{i+1}})$ is the edge weight between v_{f_i} and $v_{f_{i+1}}$. A dynamic programming algorithm can be applied to find the global optimal solution efficiently. The major issue with this method is its undesired bias toward more segmentation boundaries, *i.e.*, larger m, which may result in an oversegmentation of the text. This can be illustrated by a simple example in Fig. 3, where the text segments between S_1 and S_3 have a large character likeliness ($w_{13} = 1$), but this dynamic programming based method may still choose a segmentation boundary S_2 between S_1 and S_3 because $w_{12} + w_{23} > w_{13}$.

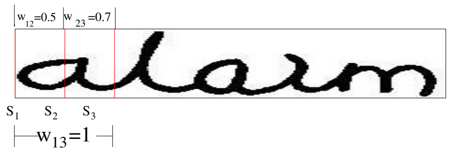

Figure 3. An illustration of the problem of dynamic programming.

In this paper, we propose to address this problem by defining a new cost function

$$\begin{aligned}&(m^*, f_i^*; i = 1, 2, \cdots, m^*)\\ =\ &\arg \max_{m, f_i; i=1,2,\cdots,m} \frac{\sum_{i=1}^{m-1} w_{f_i, f_{i+1}}}{m-1}.\end{aligned}$$

which finds a path between v_1 and v_n with the maximum average total weight. In this paper, we call this path the *average* longest path. Clearly, by introducing the path length in the denominator in Eq. (1), we remove the bias toward a larger m and avoid oversegmentation of the text. In the following, we show that the average longest path in the constructed graph G can be found in polynomial time, and present such a polynomial time average longest-path algorithm.

First, on the graph G we define a second edge weight $l_{ij} = l(v_i, v_j) = 1$ for each (v_i, v_j). We then construct an augmented *directed* edge (v_n, v_1) that starts from v_n and ends at v_1, as shown in Fig. 4(b). We also set the weight for this augmented edge as $w(v_n, v_1) = l(v_n, v_1) = 0$. This way, finding the average longest path in G is reduced to the problem of identifying a *directed* cycle C in the augmented graph G' with a maximum cycle ratio

$$\frac{\sum_{e\in C} w(e)}{\sum_{e\in C} l(e)}.$$

We then transform the edge weight w to W by $W(e) = 1 - w(e)$ for all edges in the augmented graph G'. The problem is then reduced to finding a cycle with a minimum cycle ratio

$$\frac{\sum_{e\in C} W(e)}{\sum_{e\in C} l(e)}.$$

It is easy to prove that the desired cycle with the minimum cycle ratio is invariant to the edge weight transformation

$$W(e) \leftarrow W(e) - b \cdot l(e) \tag{1}$$

for any constant b. Clearly, there exists an optimal constant b^* such that the linear transform in Eq. (1) leads to $\sum_{e\in C} W(e) = 0$. In this case, the problem is reduced to finding a zero-weight cycle with $\sum_{e\in C} W(e) = 0$. To search for this optimal b^* and this zero-weight cycle, we can use the sequential-search algorithm [2] shown in Algorithm 1 on G'.

Algorithm 1 Sequential-search Algorithm:

```
1: Initialize $b = \max_{e\epsilon E} W(e) + 1$. We know that $b^* < b$
2: Transform the edge weights using Eq. (1) and then detect a negative cycle $C$, i.e., $\sum_{e\in C} W(e) < 0$. For the initial $b$, there must exist such a negative cycle because the current $b > b^*$. If no negative cycle is detected in a later iteration, return the cycle $C$ detected in the previous iteration as a minimum ratio cycle in the augmented graph $G'$
3: Calculate the cycle ratio $\frac{\sum_{e\in C} W(e)}{\sum_{e\in C} l(e)}$ using the original edge weights $W$ without applying the linear transformation (1), using this calculated cycle ratio as the new $b$, and then return to Step 2.
```

Negative cycle detection is a well-solved polynomial-time problem [6] and has a worst-case running time of $O(n^2 m log(n))$. Here n is the number of nodes in the

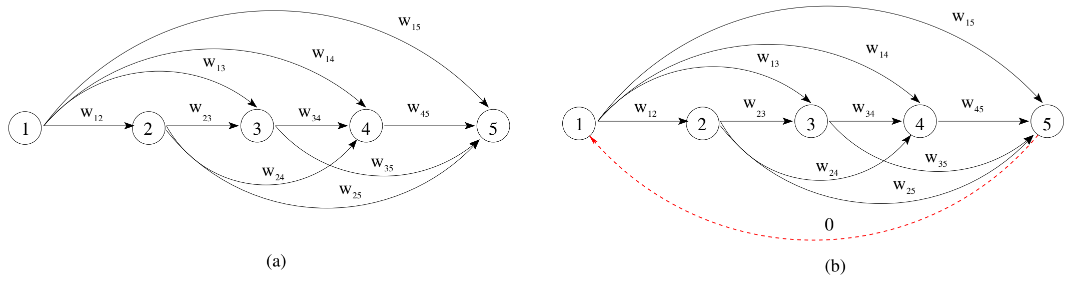

Figure 4. An illustration of adding an augmented edge for finding the average longest path in G. (a) Original graph G. (b) Graph augmented by an edge (v_n, v_1).

graph and m is the number of edges. The complete proposed handwritten text segmentation algorithm can be summarized by Algorithm 2.

Algorithm 2 Handwritten Text Segmentation:

```
1: Construct candidate segmentation boundaries by uniformly and densely sampling the text image.
2: Construct graph $G$ representing candidate segmentation boundaries as vertices. Construct forward edges between each pair of vertices.
3: Define the edge weight $w$ to reflect the character likeness.
4: Find the average longest path in the constructed graph by using the graph algorithm described in Section 2.2.
5: Keep the candidate segmentation boundaries whose corresponding vertices are included in the identified average longest path as the final text segmentation boundaries.
```

3. Experiments

In our experiments we use the standard IAM handwriting database. This database consists of 657 different writers and 1539 pages of scanned text. For testing, we randomly selected a set of 300 handwritten words from this IAM handwriting database. These words were collected from a subset of 50 different writers. Each word is made up of 2 to 9 characters, drawn from 26 lowercase characters. The characters in each word are written in a connected fashion, and we apply the proposed text segmentation method to segment each word into individual characters. We also collected characters in isolation for these 26 character classes from a training set of 200 words from the same database without any overlap to the 300 words used for testing, and we use these isolated characters as training samples for the SVM classifiers. For each character, we collected 50 training samples giving us a total of 1300 training samples.

In constructing the candidate segmentation boundaries, we uniformly sample each test word with an interval of 10 pixels. To quantitatively evaluate the performance of a text segmentation, we manually construct a ground truth segmentation for each test word. Specifically, we present, to a human evaluator, each test word overlaid by the candidate segmentation boundaries. The human evaluator simply selects a subset of these boundaries that best separate all the characters as the ground-truth segmentation boundaries. To evaluate a segmentation result, we calculate the precision and recall in finding these ground-truth segmentation boundaries. Here a text segment is true positive only if it spatially overlaps with a ground-truth segment perfectly. Table 1 shows the average precision, average recall, and their standard deviations in terms of all the 300 test words, together with the average F-score that combines precision and recall.

	AP	SP	AR	SR	F-score
Proposed	0.7968	0.2158	0.7961	0.2390	0.7965
DP [17]	0.6090	0.2203	0.5060	0.2515	0.5526

Table 1. Precision/Recall statistics for the proposed method and the comparison method on all the 300 test words. AP is average precision, AR is average recall, SP is the standard deviation of the precision, and SR is the standard deviation of the recall.

For comparison, we implement the dynamic programming (DP) based text segmentation method developed in [17]. As discussed above, this method has a bias toward oversegmentation and requires a good initial candidate segmentation with few false positives. Following the basic ideas described in [17], we adopt the following two strategies to prune more candidate segmentation boundaries before applying dynamic programming for text segmentation.

1. Prune all the candidate segmentation boundaries that enter stroke pixels more than twice when scanning from the top to the bottom.

2. Do not allow the dynamic programming algorithm to consider text segments that have an aspect ratio greater than 1.2.

For fair comparison, we use the same histogram of oriented gradients feature [13] for this comparison method as we

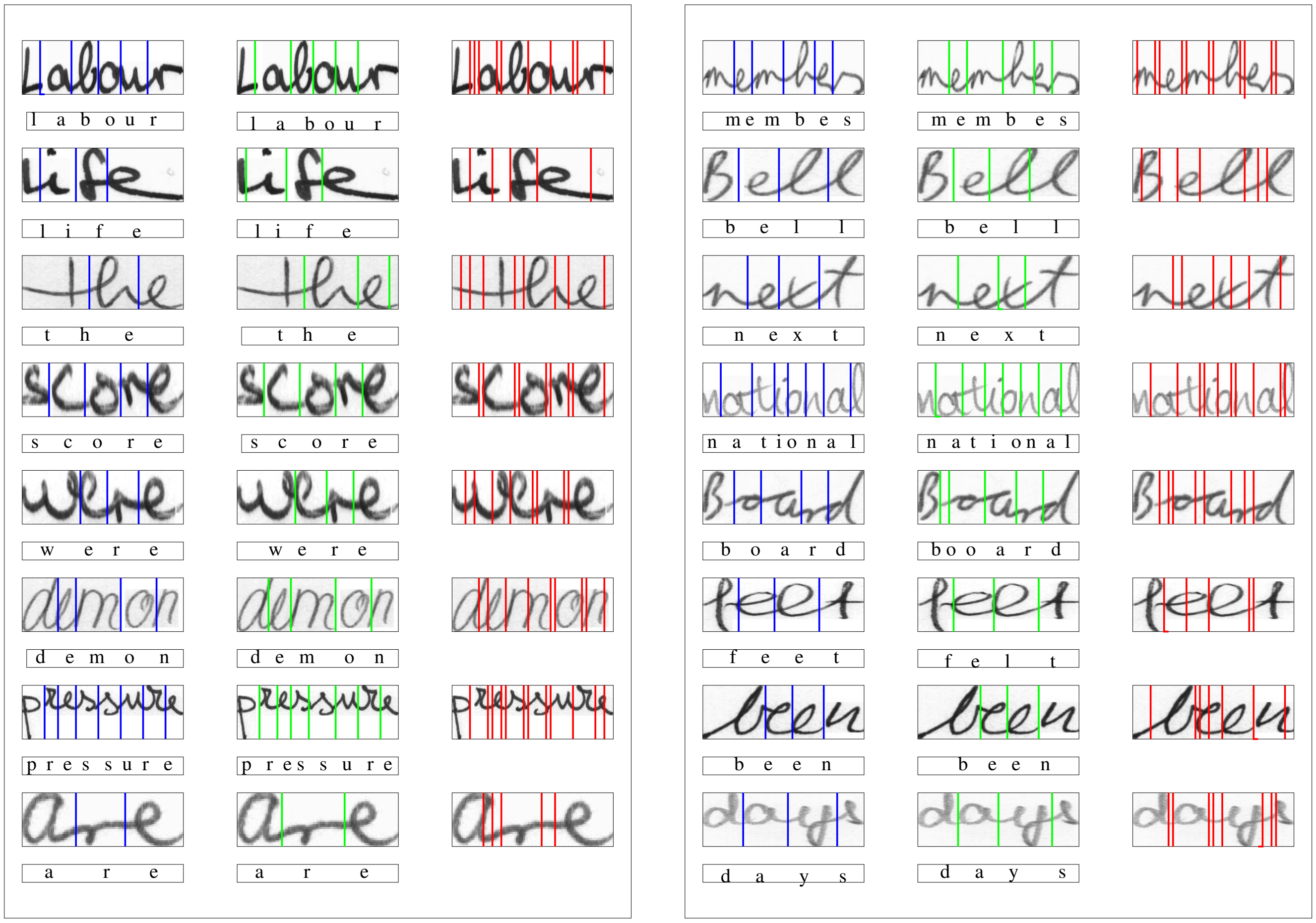

Figure 5. Text segmentation on a subset of the test words. For each test word, we show the segmentation from: (Left) the ground truth, (Middle) the proposed method, and (Right) the dynamic programming based method [17]. The characters below the test words for the proposed method (Middle column) are the recognition results from the character class corresponding to the character likeliness. The characters below the test words for ground truth (Left column) are the ground-truth characters for these words.

used for the proposed method (see Section 2). The performance of this dynamic programming method is also shown in Table 1. We can clearly see that the proposed method outperforms this dynamic programming based method. Sample segmentation results are shown in Fig. 5, where the character recognition results for the proposed method (shown below each word) are obtained from the character class corresponding to the character likeliness. We can see that, even with the additional strategies reducing false positives, the dynamic programming based method still produces many oversegmentations. Note that characters such as "L" in the word "Labour" and "B" in the word "Bell" are still recognized using the same classifier which is trained only on 26 lowercase characters.

Using the above precision/recall metric does not well quantify the segmentation discrepancy in pixels. To address this issue, we further evaluate the spatial overlap between the ground-truth segmentation and the segmentation from a test method. Specifically, given each text segment T resulting from a test method, we find its best overlapped ground-truth text segment T_G and calculate their spatial-overlap difference as

$$\phi(T, T_G) = 1 - \frac{T \cap T_G}{T \cup T_G}.$$

A lower spatial overlap difference indicates a better segmentation. We calculate the average of this overlap difference over each obtained text segment, and then over all the 300 test words. The performance of the proposed method and the comparison method, in terms of this overlap difference, is shown by the box plot in Fig. 6. Clearly, the proposed method achieves much better text segmentation.

A fair comparison with holistic methods against proposed method is not possible since holistic methods utilize trained models for each word to recognize each test word according to a fixed set of lexicon. Proposed method does not employ such a lexicon based word classifier and this will enable the use of our method to more general appli-

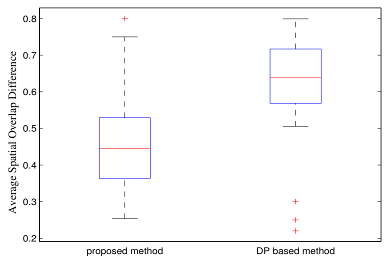

Figure 6. Average spatial overlap difference over all the 300 test words, using the proposed method and the comparison method based on dynamic programming

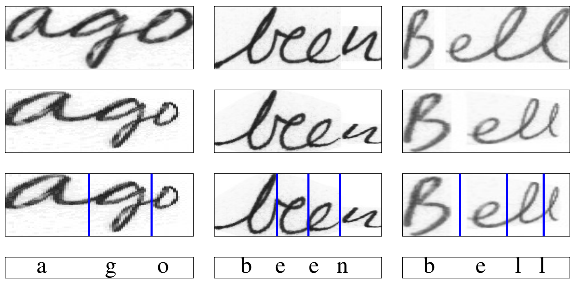

Figure 7. Examples of segmenting three distorted texts. (Top) Original texts. (Second row) Distorted texts. (Third row) Text segmentation using the proposed method. (Bottom) Character recognition from the class corresponding to the character likeliness.

cation domains where the handwritten texts cannot be modeled by a fixed set of lexicons. Since the 300 IAM handwritten texts used in our experiments are valid English words, we conduct a simple experiment to see how many words out of these 300 can be correctly recognized by applying the spelling-check tool (in the Microsoft Office) to the segmentation and character recognition results from the proposed method. If any one of the top four candidate words provided by the spelling check is correct, we count this text as correctly recognized. We found that using this spelling check, we can get 65% of these 300 test words correctly recognized. One previous holistic method [19] reported 73.45% of recognition rate on 300 test words from IAM database. However, it is unclear to us which 300 test words are used in [19] and what kind of the lexicons are used in achieving this rate.

Without setting any limiting constraints on the size and aspect ratio of the segmented characters, the proposed method has an advantage in segmenting handwritten texts in which character size varies substantially from one to another. This is particularly useful in applications where the text is distorted in scanning. For example, when the page to be scanned is not tightly pressed on the scanner, some texts may become smaller and thinner in the scanned text image. Figure 7 shows three such examples, where the right side of each word is condensed horizontally and the characters become thinner from left to right in each word. We found that the proposed method can successfully segment these words as shown in Fig. 7, even without training the SVM classifiers using any such distorted characters.

We also conduct experiments to show the effectiveness of character likeliness, as defined by the output of multiple binary SVM classifiers (see Section 2), in distinguishing characters and non-characters. In the test words, we randomly select 400 text segments, in which 200 are characters from the ground truth, and 200 are non-characters constructed by either taking the left or right half of a character, or merging the right half of one character to the left half of the next character in the same word. We evaluate their character likeliness, which is shown in Fig. 8. We can see that the 200 characters (blue) show a much higher average character likeliness than the 200 non-characters (red). However, we can also see that some characters show low character likeliness and some non-characters show high character likeliness, because of the ambiguity and complexity underlying the handwriting. For example, the left half of the character "W" is indistinguishable from the character "V" and has a high character likeliness. However, in Fig. 8 we always count half of a character as a non-character.

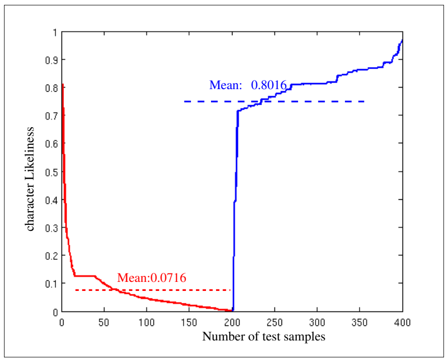

Figure 8. Character likeliness calculated for 200 characters and 200 non-characters.

This kind of ambiguity can lead to failures of the proposed method in text segmentation. Several examples are shown in Fig. 9. In Fig. 9(a), the character "w" is over-segmented into a "n" and a "v", both of which show high character likeliness. In Fig. 9(b), two contiguous characters "ub" are not correctly segmented because their combination resembles the character "w" and bears a high character likeliness. Similarly, in Fig. 9(c), two contiguous characters "ur" are not correctly segmented because their combination also resembles character "w" and bears a high character likeliness. Such ambiguity cannot be well addressed by considering only the text-image information or the stroke shape.

On the speed of the proposed method, we implemented

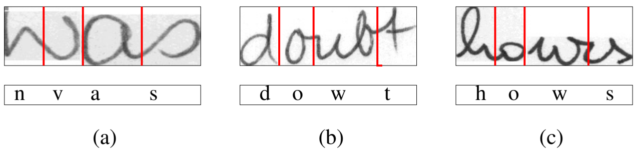

Figure 9. Examples where the proposed text segmentation method fails.

the entire algorithm in Matlab, and run on a 2GHz Linux workstation with 8 GB of RAM. For a text image with roughly 30 candidate segmentation boundaries, the proposed method takes about 10 seconds to complete. For the 300 test words, the proposed method takes an average of 14 seconds per word for text segmentation.

4. Conclusion

In this paper, we developed a new graph-based method to segment connected handwritten text into individual characters for text recognition. Different from previous methods, the proposed method does not require a good initial set of candidate segmentation boundaries, and does not make any limiting assumptions on the number, size, width, height, or aspect ratio of the characters. We adapted a character recognition algorithm using SVM classifiers to measure the character likeliness of each text segment, and then searched for a text segmentation that leads to a maximum average character likeliness. This avoids any possible bias toward an over-segmentation that encumbers previous methods. Specifically, we developed a graph algorithm to find the optimal segmentation with the maximum average character likeliness. We collected a set of real, handwritten text images for both training and testing, and found that the performance of the proposed method is superior to a previous method that uses dynamic programming.

Acknowledgements

This work was supported, in part, by AFOSR FA9550-11-1-0327, NSF IIS-0951754, NSF IIS-1017199, NEH HK-50032-12, and ARL under Cooperative Agreement Number W911NF-10-2-0060.

References

[1] Z. A. Aghbari and S. Brook. Hah manuscripts: A holistic paradigm for classifying and retrieving historical arabic handwritten documents. *Expert Systems with Applications*, 36(8):10942 – 10951, 2009.

[2] R. Ahuja, T. Magnanti, and J. Orlin. Network flows: Theory, algorithms, and applications. In *Prentice Hall*, 1993.

[3] N. Arica and F. Yarman-Vural. Optical character recognition for cursive handwriting. *IEEE PAMI*, 24(6):801 –813, June 2002.

[4] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001.

[5] T. Chang and S. Chen. Character segmentation using convex-hull techniques. *International Journal of Pattern Recognition and Artificial Intelligence*, 13(6):833, September 1999.

[6] B. V. Cherkassky and A. V. Goldberg. Negative-cycle detection algorithms. In *Proceedings of the Fourth Annual European Symposium on Algorithms*, pages 349–363, London, UK, 1996.

[7] A. Djematene, B. Taconet, and A. Zahour. A geometrical method for printed and handwritten berber character recognition. In *ICDAR*, pages 564–567, 1997.

[8] V. Govindaraju and H. Xue. Fast handwriting recognition for indexing historical documents. In *Document Image Analysis for Libraries*, pages 314–320, 2004.

[9] S. Lee and Y. Kim. A new type of recurrent neural network for handwritten character recognition. In *ICDAR*, pages 01–38, 1995.

[10] S.-W. Lee, D.-J. Lee, and H.-S. Park. A new methodology for gray-scale character segmentation and recognition. *IEEE PAMI*, 18:1045–1050, 1996.

[11] S. Liang, M. Ahmadi, and M. Shridhar. Segmentation of touching characters in printed document recognition. In *ICDAR*, pages 569–572, 1993.

[12] Y. Lu. On the segmentation of touching characters. In *ICDAR*, pages 440–443, 1993.

[13] S. Maji and J. Malik. Fast and accurate digit classification. Technical Report UCB/EECS-2009-159, EECS Department, University of California, Berkeley, Nov 2009.

[14] G. Martin, M. Rashid, and J. A. Pittman. Integrated segmentation and recognition through exhaustive scans or learned saccadic jumps. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):831–847, 1993.

[15] L. Mico and J. Oncina. Comparison of fast nearest neighbor classifiers for handwritten character recognition. *Pattern Recognition Letters*, 19(3-4):351–356, March 1998.

[16] J. Rocha and T. Pavlidis. Character recognition without segmentation. *IEEE PAMI*, 17(9):903–909, 1995.

[17] P. P. Roy, U. Pal, J. Llads, and M. Delalandre. Multi-oriented and multi-sized touching character segmentation using dynamic programming. In *ICDAR*, pages 11–15, 2009.

[18] J. Song, Z. Li, M. Lyu, and S. Cai. Recognition of merged characters based on forepart prediction, necessity-sufficiency matching, and character-adaptive masking. *IEEE Transactions on Systems,Man and Cybernetics*, 35(1):2–11, February 2005.

[19] U. v. Marti and H. Bunke. Text line segmentation and word recognition in a system for general writer independent handwriting recognition. In *ICDAR*, pages 159–163, 2001.

[20] J. Wang and J. Jean. Segmentation of merged characters by neural networks and shortest-path. In *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, pages 762–769, New York, NY, USA, 1993.

[21] X. Wang, V. Govindaraju, and S. Srihari. Holistic recognition of handwritten character pairs. *Pattern Recognition*, 33(12):1967 – 1973, 2000.

[22] H. Xue and V. Govindaraju. On the dependence of handwritten word recognizers on lexicons. *IEEE PAMI*, 24(12):1553–1564, December 2002.