2006-04-01

Separating Lines of Text in Free-Form Handwritten Historical Documents

William A. Barrett
william_barrett@byu.edu

Douglas J. Kennard

Original Publication Citation

Douglas J. Kennard and William A. Barrett, "Separating Lines of Text in Free-Form Hand-written Historical Documents," IEEE Proceedings, 2nd International Conference on Document Image Analysis for Libraries (DIAL 26), pp. 12-23, Lyon, France, April, 26.

Separating Lines of Text in Free-Form Handwritten Historical Documents

Douglas J. Kennard, William A. Barrett
Department of Computer Science
Brigham Young University
Provo, UT 84602 USA
(kennard@cs.byu.edu, barrett@cs.byu.edu)

Abstract

We present an approach to finding (and separating) lines of text in free-form handwritten historical document images. After preprocessing, our method uses the count of foreground/background transitions in a binarized image to determine areas of the document that are likely to be text lines. Alternatively, an Adaptive Local Connectivity Map (ALCM) found in the literature can be used for this step of the process. We then use a min-cut/max-flow graph cut algorithm to split up text areas that appear to encompass more than one line of text. After removing text lines containing relatively little text information (or merging them with nearby text lines), we create output images for each line. A grayscale output image is created, as well as a special mask image containing both the foreground and information flagging ambiguous pixels. Foreground pixels that belong to other text lines are removed from the output images to provide cleaner line images useful for further processing. While some refinement is still necessary, the result of early experimentation with our method is encouraging.

1. Introduction

Vast amounts of valuable historical and genealogical information exists within free-form handwritten documents such as letters, diaries, and wills. In order for that information to be readily accessible to the masses, those documents need to be transcribed or indexed (annotated) so that they are searchable.

Handwriting recognition tasks, including indexing as well as automatic transcription, require the handwritten text to be localized within images before recognition can be performed. In many applications, such as reading check amounts, postal addresses, or forms with a predictable layout, it may be relatively easy to locate and separate the handwritten text due to the constraints of the layout in those specific domains. However, many historical documents (such as letters, diaries, and wills) have a "free-form" layout, in which the handwritten sentences flow across the page in left to right, top to bottom fashion.

Due to the nature of free-form handwriting, it is difficult to reliably localize and separate the text in a general manner. Lines of text are not always exactly parallel to each other—sometimes they are even slightly curved— and the spacing between the lines often varies somewhat from one part of the page to another. In addition, lines of text may touch each other due to the ascenders or descenders of characters from one line overlapping with those from another.

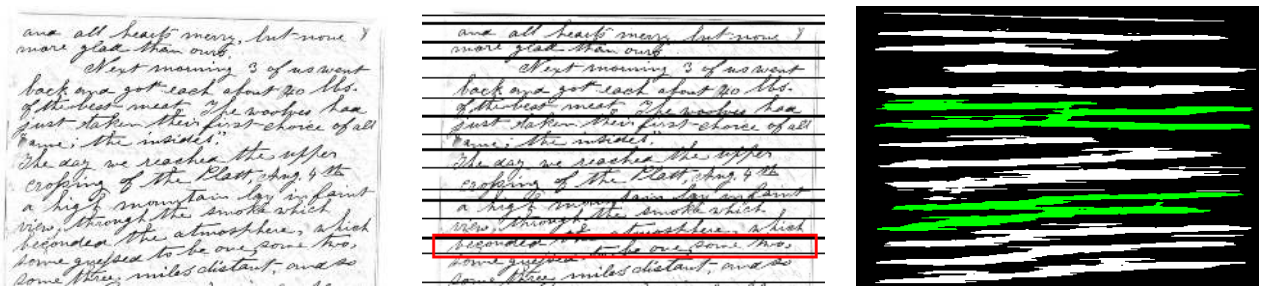
When the documents of interest are historical documents, the difficulties are exacerbated by the problems that may arise due to degradation and variation in the documents, themselves. In addition to the noise, skew, distortion, and uneven lighting that is often introduced into an image during microfilming or digitization, the original document is often already plagued with smudges, smears, faded print, and an uneven or discolored background, before the document is ever digitized. Bleed-through (or shine through) of writing from the opposite side of a document is also common in historical documents, and when severe, can make it difficult to automatically distinguish the actual text from the bleed-through. In addition, many handwritten documents are written on paper that has rules and lines that intersect or overlap the handwriting, interfering with and complicating the process of word separation.

While various authors have addressed the topic of separating text lines and words, there remains a need to develop even more robust methods that can handle the difficult problems that are so often encountered in historical handwritten documents.

In [14], by Senior and Robinson, the process of locating lines of text is based solely on the gaps between lines, thereby making the assumption that lines are well separated.

Yanikoglu and Sandon [18] use projection profiles to determine where the boundary between the baseline of a line of text should be and the half-line of the line of text below it. They then apply a contour following algorithm in that

Proceedings of the Second International Conference on Document Image Analysis for Libraries (DIAL'06)
0-7695-2531-8/06 $20.00 © 2006 IEEE



(a) (b) (c)

Figure 1. Correctly locating lines of text is difficult on this image. a) Original preprocessed image. b) Black lines show where a profile-based method breaks the lines of text. The highlighted region shows one of several lines that are handled incorrectly. c) A binarized ALCM before postprocessing. The highlighted connected components each encompass more than one line of text.

boundary area.

Kavallieratou, et al. [6] also use projection profile methods to determine the number of lines and estimate where lines begin. The area of the image corresponding to the space between lines is then examined to trace the boundary between text lines. The trace of the line boundary moves up or down as needed to go around ascenders or descenders that get in the way, and follows the profile minimum when there is no clear path due to ascenders/descenders that go all the way from one line to the other.

Nicolas, et al. [11] use an approach based on the Artificial Intelligence concept of production systems in order to search for an optimal alignment of connected components into text lines. The results and conclusions of the paper show that more work is required before this method will work robustly on difficult historical documents.

Manmatha, et al. [9] use a scale space technique for word separation that produces good results on a large collection of George Washington's manuscripts, as well as some other documents. Segmentation of text lines is performed using smoothed projection profiles, which typically is sufficient for the documents used in the tests, since the lines are relatively straight. For some historical documents, however, profile methods will not suffice due to curvature in the lines of text, as shown in Figure 1.

Zhixin Shi, et al. [16] use an Adaptive Local Connectivity Map (ALCM), in which the value at each pixel is the sum of all pixels from the original image within a specified horizontal distance of that pixel. The ALCM is then thresholded using Otsu's Method [12], and the connected components represent probable regions for lines of text, or partial lines of text. Some postprocessing is done to remove small, redundant components. The method handles slight curvature in lines that profile methods do not handle. However, this method can still have problems when the characteristics of the particular handwriting or document being processed cause the resulting segmentation to have multiple lines of text encompassed within a single connected component, as shown in Figure 1.

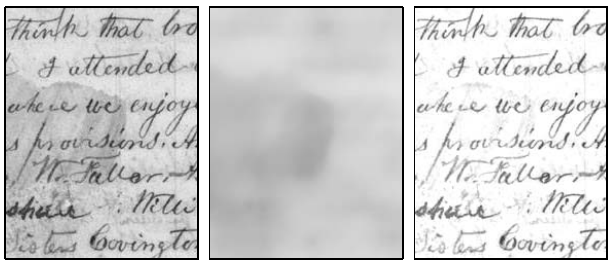
In this paper, we present a novel method for locating lines within free-form handwritten historical documents. Our method uses an approach to find initial text line candidates that, although distinct, somewhat resembles the ALCM method described in [16]. Alternatively, the ALCM could be used for this step, as we discuss. An additional post-processing step checks for text line components that are likely to contain more than one line of handwritten text, and splits these components using a min-cut/max-flow algorithm described by Boykov and Kolmogorov in [3]. The splitting process is repeated until no more components appear likely to encompass multiple lines of text. After merging or removing lines with little information, we then create output line images.

In Section 2, we discuss our current preprocessing methods. In Section 3, we present our algorithm for locating text lines, merging redundant components, and splitting components that encompass multiple lines of text. In Section 4, we report results of early tests, and we discuss our conclusions in Section 5. Possible future work is discussed in Section 6.

2. Preprocessing

We preprocess document images to reduce the number of artifacts that would interfere with text line separation, word separation, and subsequent handwriting recognition tasks. Our current preprocessing consists of the following steps:

- Background Removal
- Page Deskewing
- Global Threshold Selection
- Rule/Margin Line Removal



(a) (b) (c)

Figure 2. Background removal using median filter. a) Portion of original image. b) Background image obtained using circular kernel with 18-pixel radius. c) Image after removing background and stretching histogram.

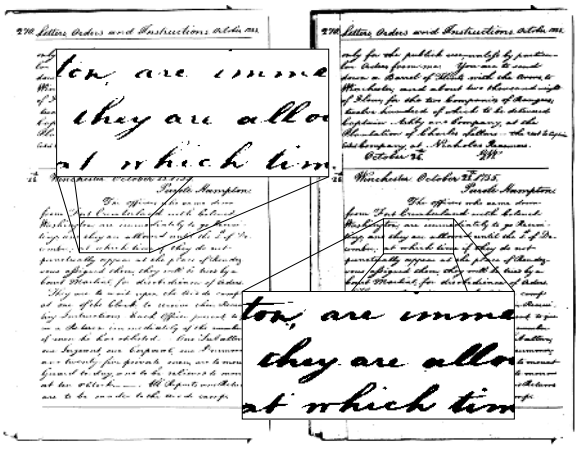


Figure 3. Automatic threshold selection (after background removal) for an image from the collection of George Washington manuscripts used in [7]. a) Otsu's Method - notice that words and letters are made up of many disjoint components. b) Our Method - notice that letters and words are connected better.

2.1. Background Removal

Background removal is performed using the method described in [4, 5]. In this method, an image is median filtered with a circular kernel to remove the foreground elements (such as handwriting). This, in effect, produces a background image that can be subtracted from the original to produce a foreground image. A histogram stretching operation normalizes the background intensity of the resulting foreground image to white.

As shown in Figure 2, the resulting image contains the foreground of the original image (including text and other small features), but without the variations in background intensity or large blobs. The dark margin areas that often appear at the edges of images are usually removed by this step, as well.

Color images are converted to grayscale before background removal since we do not currently take advantage of any additional information that could be gained from the color signal.

2.2. Page Deskewing

Similar to many other deskewing techniques, our method is based on projection profiles of the images, taken at various angles. Initially, we use a vertical Sobel filter to accentuate the tops and bottoms of the text. Then, for each angle, we compute a slanted projection profile of the pixel values of the grayscale edge image, normalizing each profile value by the number of pixels represented in that profile location. The variance of the profile is calculated, and the angle with the maximum variance is assumed to be the skew angle. To reduce computation, we first find the best skew angle estimate at a coarse resolution of angles, then progressively refine the estimate using higher resolution around the coarse estimate. The idea of finding a course skew angle estimate before refining the angle is found elsewhere in the literature (see [6], for example).

While individual lines of text may still be curved or skewed, globally deskewing the document serves to at least bring them close to horizontal to facilitate the text line location and separation, which assumes that the text lines will be fairly horizontal. It also should bring rule lines and lines at the edges of the page closer to their respective horizontal or vertical orientations, making line removal easier.

2.3. Global Threshold Selection

We create a binarized version of our background-removed, deskewed image because several of the steps in our system require a binarized image. Since background variations of the document are already removed, a global threshold suffices for binarization in most cases. Automatic selection of a proper threshold, however, can be tricky.

Otsu's [12] algorithm for automatic threshold selection is widely used in many applications. However, we find that the Otsu method often selects a threshold that is lower than we desire for our purposes, especially when the handwritten ink does not appear consistently dark on the page. On strokes that are faint, the ink rises above the threshold, and will not appear in the binary image. This causes words or sequences of characters that should be connected to become disjoint components in the binary image (see Figure 3).

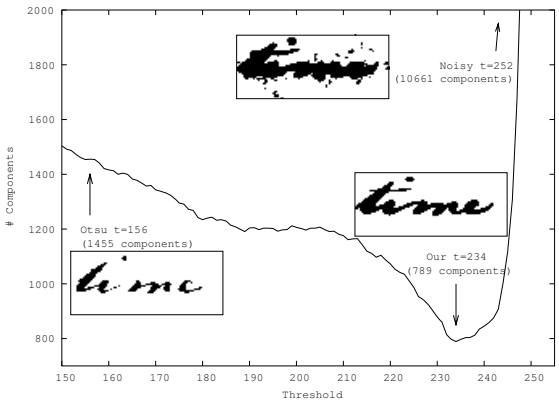


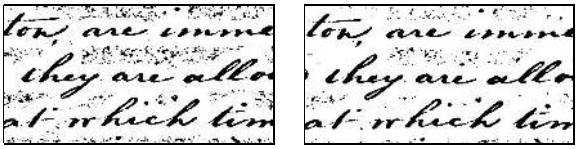
Figure 4. Graph of t_i vs number of connected components, and portion of image shown at 3 different t_i values: Otsu (faint), ours, and too high(noise)

We use the Otsu method to choose an initial threshold value, t_0. We then choose our threshold value, t, to be the value between t_0 and 255 (the maximum grayscale value) that minimizes the number of connected components in the binary image image resulting from thresholding at that value.

Due to the nature of handwritten document images, the automatically selected value of t tends to strike a good balance between maximizing the "connectedness" of handwritten words and reducing extraneous noise, as shown in Figure 4.

In some extreme cases, such as when there is severe bleed-through between the text lines, connected components that should stay separate can "merge" as the threshold value is increased. Even though the higher threshold causes more noise (bleed-through) to appear, the number of components at the higher threshold is lower due to the fact that the noise being introduced touches what would be multiple components at the lower threshold. The number of components can continue to decrease as the value of t becomes higher, until almost the entire page becomes one big component. In order to prevent this erroneous selection of an excessively high threshold, we limit our choices to those values of t in which the number of pixels belonging to any given component does not exceed the total number of pixels that belonged to all components when t_0 was used.

While the advantage of our global thresholding method is not always as obvious as the example used in Figures 3 and 4, we empirically find that our method usually works noticeably better than simply using the Otsu method for the images in our current test set. Only in some rare cases with severe bleed-through do we see our method perform slightly worse.



(a) (b)

Figure 5. Compare our thresholding method in Figure 3 to Niblack adaptive thresholding. a) Niblack applied to original grayscale image (no background removal). b) Niblack applied to image after background removal.

For comparison with our method, Figure 5 shows an example of Niblack's adaptive thresholding [10] applied on the image before background removal, or applied after background removal. As can be seen, the letters are more disjoint than our method, and noise is exaggerated in areas distant from the foreground text.

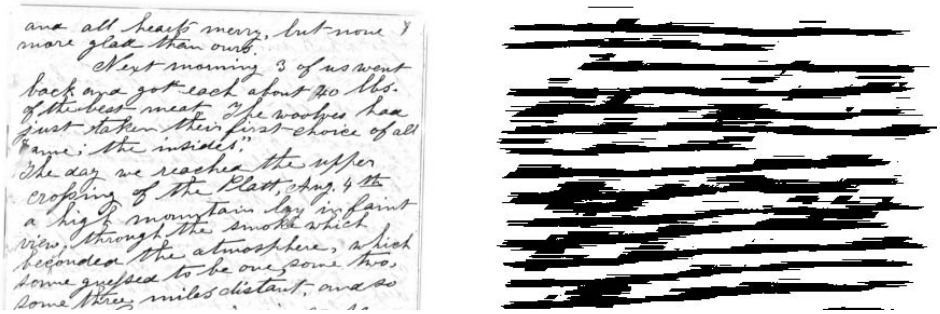
2.4. Rule/Margin Line Removal

Line removal is performed to suppress lines (such as horizontal rule lines and vertical margin lines) that would likely interfere with separation of words and lines of text, as well as subsequent recognition tasks. Various line removal techniques exist in the literature. We choose to perform line removal based on run-lengths of foreground pixels in the binarized image.

For vertical line removal, (nearly) vertical runs of foreground pixels exceeding a particular length (currently 1.5 times the median line spacing estimate from Section 3.1) are considered to belong to vertical lines. Foreground pixels within a very small neighborhood of the vertical runs, as well as pixels with high derivative magnitude (pixels which are likely to be the line edges blurred with the light background) are also considered to be vertical line pixels. The line pixels are removed by linearly interpolating the intensity of the pixels to either side of the line.

Horizontal line removal is similar to vertical line removal, but the run-length threshold is larger because we do not want long ligatures, "T"-crossings, etc. to be mistakenly removed.

Our line removal algorithm works relatively well when the lines show up clearly in the binarized image. If lines are so faint that they don't show up in the binarized image, or show up as short, disjoint segments instead of continuous runs of pixels, then our algorithm does not detect and remove the lines properly. Thin lines are removed inconspicuously, but thick lines tend to leave noticeable smears when they are directly adjacent to handwriting, due to using interpolation to fill in where the lines were.



(a) (b)

Figure 6. Line locations determined using transition information. a) Original preprocessed grayscale image. b) Binarized transition count map before postprocessing.

3. Text Line Segmentation

The core of our approach for finding and separating lines of text in a handwritten document consists of five main parts. First, we estimate the spacing and height of text lines,as described in Section 3.1. Second, a black/white transition count map is calculated and binarized as described in Section 3.2. Third, the connected components in the resulting binarized map are analyzed, and those which appear to contain more than one line of text are split using a min-cut/max-flow graph cutting algorithm, as described in Section 3.3. Fourth, line components that encompass relatively small amounts of character data are merged with nearby line components, as described in Section 3.4. Last, a grayscale image of each text line is created, along with a mask of foreground text and ambiguous components. This is described in Section 3.5.

3.1. Parameter Estimation

In order to perform some steps of our approach, we use an estimate of the height (or thickness) of the lines we are dealing with as a parameter. This allows us to automatically choose the size of window to use when calculating the black/white transition count map, as well as parameters used in the splitting and merging of components, for example. We estimate the median text line thickness and spacing by analyzing the peaks of the (smoothed) horizontal projection profile of the preprocessed grayscale image.

3.2. Text Line Location from Transition Information

Using the global thresholding method described in Section 2.3, we create a binarized version, $f_b(x, y)$, of the preprocessed image. We then create a black/white transition count map, $M(x, y)$, in which the value of each pixel of the map is set to the number of transitions (from white to black or black to white) in the binary image that occur within a horizontal window centered at that pixel location. Specifically, using a window size of $2d+1$ pixels, the values of the transition count map are calculated as:

$$M(x, y) = \sum_{j=x-d}^{x+d} m(j, y),$$

where

$$m(j, y) = \begin{cases} 0 & \text{if } f_b(j-1, y) = f_b(j, y) \\ 1 & \text{if } f_b(j-1, y) \neq f_b(j, y). \end{cases}$$

Like the calculation of the ALCM in [16], the transition count map can be implemented using a sliding window for better efficiency. To define the window size used in creating the map, we currently use $d = 3k$, where k is the median line thickness from Section 3.1.

We use Otsu's Method [12] to binarize the transition count map, which results in a set of connected components representing the likely locations of text lines, as shown in Figure 6. We then remove very small components (those whose height or width fall below a particular threshold), and those which encompass few foreground pixels from the binarized transition count map.

Transition count information is also used elsewhere in the literature. For example, in [1], transition counts (referred to as "crossing counts") are used as a feature to differentiate the type of content in a region of a document image (e.g. text or graphics), and also as a stopping condition to avoid over-segmenting lines of printed text. We are unaware, however, of any previous work that directly uses a map of transition counts to locate lines of text.

In practice, we find that the binarized transition count map and the binarized ALCM are quite similar in many respects, but each has strengths and weaknesses that we discuss further in Section 4. The binarized ALCM, calculated directly from the preprocessed grayscale image, could be used instead of a binarized transition count map with no other significant changes to our overall approach for segmenting text lines.

3.3. Splitting Lines using Graph Cut

Components that are likely to include more than one line of text are detected by analyzing how many pixels "internal" to the component are not actually part of the component. The pixel at (x, y) is internal to the component if there is at least one component pixel somewhere above the pixel and also at least one component pixel somewhere below the pixel. More formally, the pixel (x, y) is *internal* to component C iff $\exists y_1, y_2$ where $(x, y_1) \in C$ AND $(x, y_2) \in C$ AND $y_1 < y < y_2$. Components that include more than one line of text are assumed to have a higher percentage of internal pixels that do not belong to the component (due to the spacing between the text lines) than components that only include one line of text.

To split components that have more than one text line, we use a min-cut/max-flow graph cut implementation that was made available by Kolmogorov for research use [3, 2]. Given a directed, weighted graph with two special terminal nodes called the "source" (labeled s) and the "sink" (labeled t), the algorithm "cuts" the graph (removes edges) so that s and t end up in disjoint subgraphs. The cut is made so as to minimize the sum of the weights of the removed edges.

Normal edges, referred to as "n-links" connect normal nodes in the graph, and their weights represent the cost of removing that edge. Special "t-links" connect regular nodes with the terminal nodes, and define the cost of assigning the pixels to that side of the graph (see Figure 7).

In our graph, each pixel belonging to the component is represented by one graph node. When the graph is cut, the assignment of a node to the source or the sink subgraph is analogous to assigning the corresponding pixel either to the current component or to a new text line component.

Before the graph is cut, we make a list of pixels that are likely to belong to the current component's text line, and call them the source candidates. We also make a list of the pixels that should be in the new component, called the sink candidates. These are the uppermost and lowermost component pixels, respectively, of each pixel column that has a large number of non-component internal pixels.

We form t-links between the candidate pixels and the terminals. We set up a single n-link between each pair of nodes that represent neighboring pixels. We want vertically-aligned pixels to split much more easily than those aligned horizontally, since pixels next to each other are likely to be part of the same line. For this reason, we set the edge cost between vertical neighbors to 1, diagonal neighbors to 2, and horizontal neighbors to 400. These values are chosen empirically, and may eventually need to be chosen as a function of image or text line size.

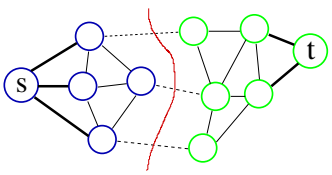


Figure 7. Min-cut/max-flow graph cut splits a graph into two subgraphs. For simplicity, we show (and use) a non-directed graph. T-links are shown thicker than n-links.

Figure 8 shows the results of using the graph cut algorithm to split text line components. After each split, the process of checking for multiple lines (and possibly doing another split) is repeated in case more than two text lines were encompassed by the same component.

One caveat of how we use the graph cut method is that it is possible to choose some erroneous source and sink candidates, which results in a bad segmentation– two or more disjoint components for at least one side of the graph. After performing a graph cut, if more than one component exists for a given side, then the top-most component is considered to be associated with the source and the bottom-most component is considered to be associated with the sink. Any candidates from the other disjoint components are discarded, and the graph cut is executed over with the reduced set of candidates. The process is repeated until a good cut with one component for each side of the graph is returned.

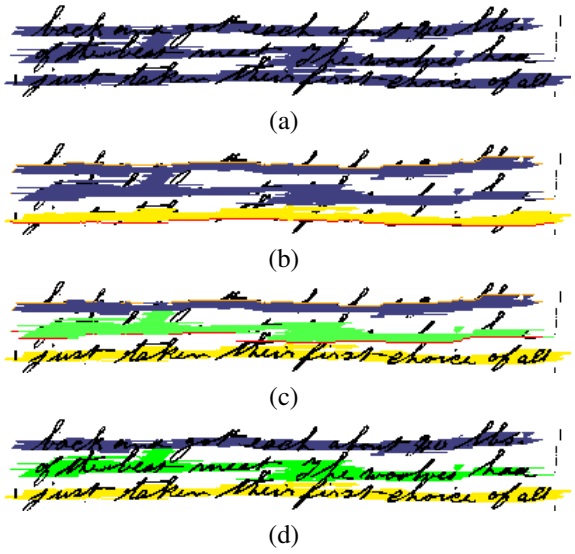
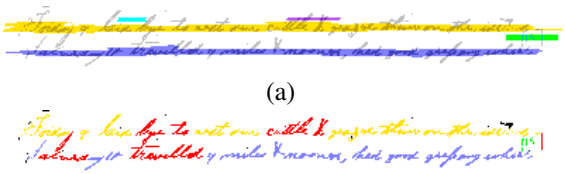
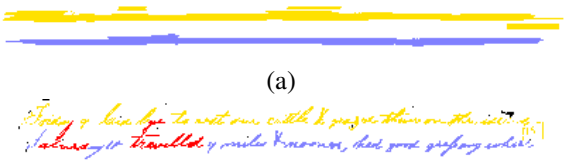


Figure 8. Splitting a component using graph cut. a) Original component. b) New component (yellow) for bottom line is formed by graph cut (light orange pixels along top of component are source candidates, dark red pixels along bottom are sink candidates). c) Graph cut used again to split the top two lines. d) Final segmentation.



(a) (b)

Figure 9. Foreground components assigned to text lines before merging. a) Text line components overlaid with foreground components. b) Foreground components colored according to their assignment to text line components. "Ambiguous" components are dark red. Black components are unassigned (not touching any text line).



(a) (b)

Figure 10. After merging the text line components from Figure 9 that contain little foreground data. a) Notice the three small components that have merged into the top line. b) Foreground components colored according to line assignment.

3.4. Merging Lines

To aid in discussion of our method, we emphasize the distinction between *text line* components and *foreground* components. Text line components (originating from the binarized transition count map, or alternatively from a binarized ALCM) are the long, thin components that represent regions of the image that encompass lines of text. Foreground components (originating from the binary version of the preprocessed document image) are the connected components that make up letters, words, and other foreground objects in the image.

Each foreground component is assigned to belong to a particular line of text if any portion of the foreground component overlaps the text line component for that line of text. If a foreground component touches more than one text line component, it is not clear which line the component should belong to, so it is considered to be an "ambiguous" component. Figure 9 shows an example of the foreground (text) components assigned to their respective text lines, with ambiguous components shown in red.

The foreground text *pixel area* for each text line component is calculated as the total number of pixels belonging to all foreground components assigned to that line, plus the number of pixels within the boundary of the text line component that belong to "ambiguous" components.

Text line components that have a relatively small pixel area (compared to the pixel area of typical text line components in the image) become candidates to be merged with nearby text line components if they are spatially close enough. The candidates are merged with neighboring components only if the resulting merged component will not be too "tall" relative to the typical thickness of the text line components in the image, or if the height of the line will not be increased. When candidates are merged, they remain spatially disjoint, but the component being merged is relabeled to match the component it is being merged into, as show in Figure 10.

Merging small text line components in this manner reduces the number of false text lines. At the same time, the amount of ambiguous foreground text is often reduced since some previously ambiguous foreground components may only touch a single text line component after the merger. For example, compare the top line of text before and after merging lines in Figures 9 and 10, respectively.

In preliminary tests, we find that our method of line merging works quite well in most cases, however, it relies on a "one size fits all" parameterization to determine both whether a line should be merged or not, and if so, which line it should be merged to. In some cases, components that should be merged are not because they are far enough away from their nearest neighbor or because they would make the line too tall if they were merged. In other cases, components may be merged that should not be due to the fact that they are very close to another component and do not cross the threshold of being too tall. Components may also be merged with the wrong line, since the closest component is not always the component that it should be merged with, especially if the author writes lines close together or uses long ascenders and descenders.

3.5. Mask and Line Image Creation

The final step in our line segmentation approach is to create text line images for output. We create both a grayscale version of the line image and a special mask of the line image. The mask includes the foreground pixels of the line image in addition to specifying which pixels may not really be part of the line because they are ambiguous or unassigned.

Since the text line components tend to cover mostly the area of the base of characters, with ascenders and descenders protruding from the line components, we first expand the line component regions by morphological dilation repeated k times, where k is the median line thickness from

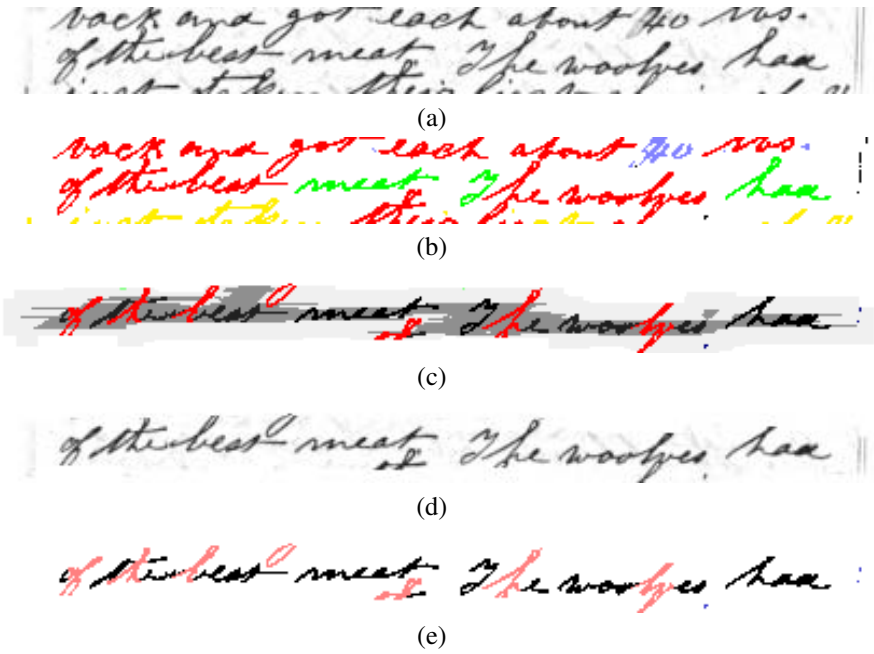


Figure 11. Output Line image creation. a) Original preprocessed grayscale image. b) Original component line assignments (ambiguous components are dark red) c) Region of interest (light gray) and text line component (dark gray) overlaid with foreground text. Pixels that remain ambiguous are lighter red. d) Output grayscale line image. e) Output line mask (ambiguous components are red, unassigned components are dark blue).

Section 3.1. As the region is expanded, the result of the dilation is constrained such that the expanded line component will not overlap any of the other unexpanded line components. Any spaces between the top-most and bottom-most pixel in each column of the expanded line component are then filled in to eliminate horizontal streaks of whitespace that protrude into the component. This expanded line component defines the *region of interest* for the line of handwritten text.

The grayscale values within the region of interest from the original preprocessed grayscale image are copied into the grayscale line image, and the foreground components within the region are copied into the special mask image. Any foreground components within the region of interest that are assigned to other lines are eliminated from the mask, and are also removed from the grayscale line image by setting the pixels of those components (and also their 4-connected neighbor pixels) to the background color. Any "ambiguous" foreground components that do not touch the original line component area are eliminated in like manner, as are unassigned components that are very far below the original line component, since these are all assumed to be either noise or ascenders/descenders from other text lines.

We take notice of the fact that in many cases, an ascender or descender of a word touches part of another line, causing the entire word to be considered ambiguous because of the fact that the whole word is a single component. Therefore, in our output mask image, we mark all ambiguous pixels as part of the line except for those portions of the ambiguous component that can directly be marked by sweeping upward or downward from the edges of the region, within a bounded angle. That is, as we sweep down from the region edge, any pixel that has a pixel marked as ambiguous directly above it or on either diagonal above it will continue to be marked ambiguous. To account for the forward slant of most writers, we also check the pixel to the right of the top-right diagonal. The process is similar sweeping up from the bottom. Figure 11 shows an example of text line output images created using our approach. In this example, it is also easy to see that the original component assignment had many entire word components marked ambiguous, but only the areas immediately connected to the edges of the region of interest remain ambiguous in the final mask.

Line images are discarded if the ratio f/c (where f is the number of pixel columns containing unambiguous foreground pixels, and c is the total number of pixel columns containing the text line component pixels) is less than a certain threshold (currently 0.25). This helps reduce the number of spurious line images that are generated due to small line components that fail to merge properly during the line merging process described in Section 3.4.

The final output images of our line separation approach

could be used either directly as input to other systems (if the system requires a line of text), or after further processing to separate words within the lines. Word separation could be performed using existing methods, such as the scale-space approach [9] or gap metrics [8, 13], applied to the output line images from our system. Systems that cannot handle grayscale could be given a binary version of our mask image, easily created by setting all non-white pixels to black.

The purpose of providing a separate mask image is to allow flexibility for future recognition systems. A recognizer that is unable to recognize a word with very high confidence could look at the mask, and attempt to recognize the word again while ignoring ambiguous regions that may be caused by stray marks such as descenders from other lines. When erroneous marks are ignored, the word may be recognized with more accuracy and higher confidence.

4. Results

We compare the results of our method to the results using profile-based line separation on a small number of images (more extensive comparison is yet to be completed). The profile method is run after performing the exact same preprocessing as used in our method to make the comparison fair. For each image, a grayscale projection profile is computed, then convolved with a Gaussian derivative kernel. Positive to negative zero crossings of the derivative are used to detect the intensity peaks, which correspond to the horizontal areas of the image with the most white, representing the space between text lines. The sigma value and kernel size used to create the Gaussian derivative kernel are computed automatically as a function of image height.

Our test images for this comparison includes 20 images from the George Washington manuscripts used in [7], and 6 images downloaded from the "Trails of Hope: Overland Diaries and Letters, 1846-1869" (Trails of Hope) on-line collection of pioneer letters and diaries, made available at URL http://overlandtrails.lib.byu.edu by the Harold B. Lee Library at Brigham Young University. Several more images from this second collection are used for visual inspection, but are not yet included in our numerical reporting. Images in the Trails of Hope collection provide samples of different writing styles, penmanship, and document quality, with special effort to select some images that have characteristics making segmentation difficult.

Table 1 lists how many errors occur in the results of our line separation method and the profile-based method, as determined by manual inspection. *Split errors* represent lines of text in which part of the line was split off that should not have been. This measure is slightly subjective (and approximate) because there are many times in which minor errors occur, such as the top of a letter being cut off. In some cases, this is counted as an error (such as when the top of an "R" gets cut off, leaving it looking like a "K" instead), while it is not counted as an error if it appears there is no real impact (the last little bit of an already obvious extender, for example). *Unsplit errors* are lines of text that should be split, but aren't. *Extra Lines* are lines that are considered distinct, but should not exist – typically caused by noise in the image, especially in the top or bottom margin. *Missing Line* errors result from handwritten text that gets discarded, but that should be part of a text line.

Our Method	**Washington**	**Trails of Hope**
(Good Lines)	(662)	(146)
Split Errors	1	9
Unsplit Errors	1	4
Extra Lines	8	0
Missing Lines	5	0
Profile Method	**Washington**	**Trails of Hope**
Split Errors	18	18
Unsplit Errors	6	5
Extra Lines	46	23
Missing Lines	n/a	n/a

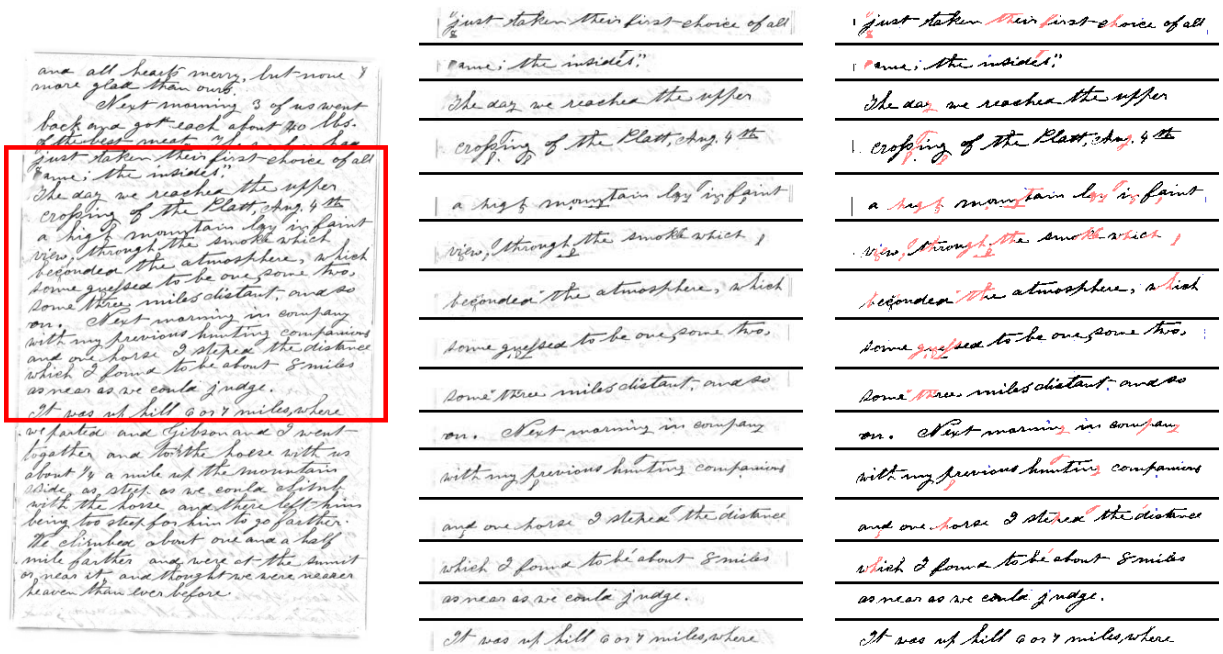
Table 1. Line Separation Error counts.

As can be seen in Table 1, both our method and the profile method perform very well on the first set of test images (the George Washington letters). The lines of text in these letters are written neatly and are well-spaced.

Our method separates 662 lines properly, including 4 that are aligned horizontally, but spaced significantly apart. Such splits are not considered errors since they would not impede later recognition tasks.

Only one split error is seen, which is a tall letter "S" in which the upper half is split off. The only unsplit error is a small, two-word phrase inserted in the space between two real lines of text, which did not get split out. This also causes an error with the profile-based method. All but one of the extra lines are caused by noise at the bottom of pages. The remaining extra line results from a round stamp or seal (non-text) on the document. All of the missing lines are extremely short– dates at the top, or short words / abbreviations that end up on their own line.

The profile method has slightly more errors. Many of the split errors are very minor (the tops of letters or the raised "th" in numbers like "10th" being split off from the line they should be in, for example). Unsplit errors seem more frequent (especially for short lines) using the profile method. In addition to the 6 we report, there are also a couple that probably should be split, but are not counted as errors because they do not overlap (one above the other), and therefore would not hinder later recognition tasks. Extra lines are all caused by noise at the top and bottom of images. We do not remove empty lines, so all portions of the image belong to exactly one line and there are no "missing line" errors.



(a) (b) (c)

Figure 12. Results of our text line separation. a) Preprocessed image. b) Grayscale Output Line Images (concatenated) from the highlighted region. c) Corresponding Output Line Mask Images.

For the few documents that we compare so far in Table 1, we see that similar results hold for the six Trails of Hope images– our method performs slightly better on average than the profile method.

It is noted that the George Washington manuscripts are well-spaced, and have a fairly consistent slope. This is the case also for many of the Trails of Hope images. We are currently in the process of collecting more images representative of the problems we set out to solve, especially images that do not have consistent slope. We anticipate that our method will perform much better than profile methods on such images. Two such images that are already included in our tests (and Table 1) are shown in Figures 12 and 13, along with the output line images from our method.

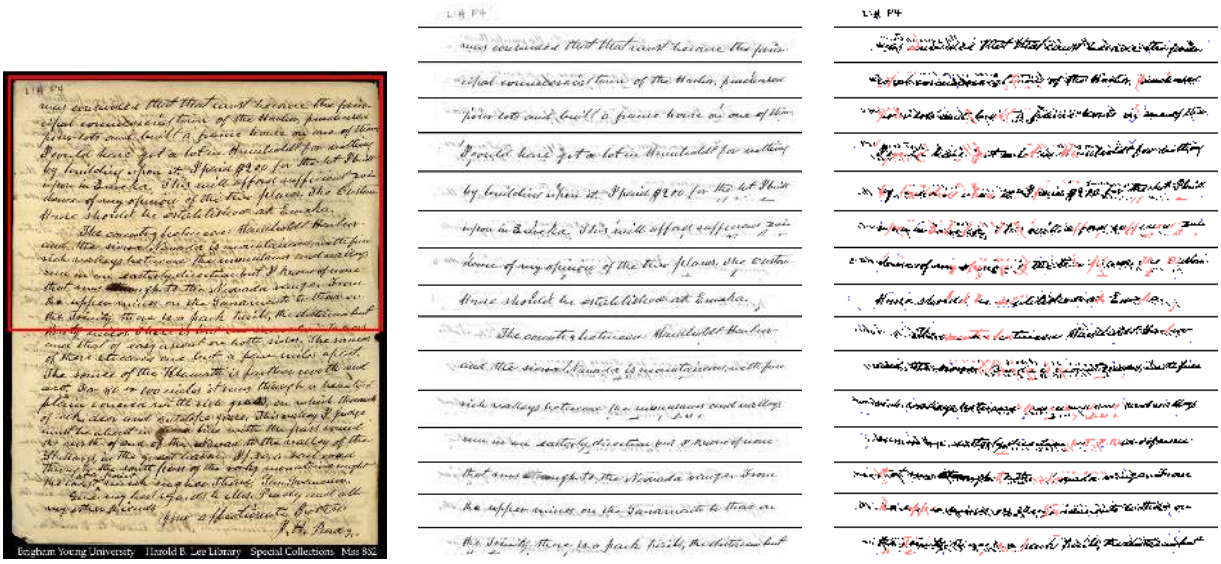
Applying our method to several other documents from the Trails of Hope collection that are not yet quantified in Table 1, we find that our method often performs quite well. In addition to successfully locating lines of text, our system is typically accurate in marking ambiguous areas of the line that may not actually belong to the line. This information could be useful to recognition systems.

As we experiment with particularly difficult documents in our data set, we see that our method does struggle in some cases, such as when there is excessive bleed-through, as can be expected from most methods. It also tends to have problems (such as the "extra line" errors reported in the comparison above) when the preprocessing (line removal, etc.) fails to clean up documents properly. Improvements to our preprocessing will help make the performance of the complete system more robust. Also, because of the fact that a horizontal window is used to either count transitions or generate an ALCM, our method will not work properly if text lines remain skewed too far away from horizontal even after the global deskewing step.

Some other cases in which our method sometimes falls short are when there is large variation in the size of writing (for example, a word or line that is much taller than is typical for the page), when lines are very close together, when many ascenders/descenders are near each other, or when lines of text are extremely short in length.

While our method does not always handle these situations perfectly, we believe that it does, on average, represent an improvement over the traditional methods we have discussed, especially for documents in which the line skew is not consistent. In addition, our method cleans up some of the stray marks in the output line images and provides information about ambiguous pixels in the line image, which traditional methods do not.

As mentioned in Section 3.2, both the black/white transition count map and the Adaptive Local Connectivity Map (ALCM) can be used to locate the likely regions of text lines. Our experiments indicate that the two methods usu-



(a) (b) (c)

Figure 13. Results of our text line separation. a) Preprocessed image. b) Grayscale Output Line Images (concatenated) from the highlighted region. c) Corresponding Output Line Mask Images.

ally give similar results, but each has strengths and weaknesses. While the transition count method does a better job of ignoring solid non-text areas, it is more susceptible to falsely detecting faint, noisy regions that are not solid. The ALCM method tends to have fewer "ambiguous" components due to the text line components being thinner vertically. On the other hand, the transition count method tends to do better at including disjoint portions of ascenders as part of the line instead of leaving them as unassigned components. The transition count method also tends to locate lines of text that are short in length better than the ALCM method.

Whichever method is used, our subsequent use of the graph cutting algorithm is usually very effective in splitting the text line components in reasonable locations so that each line component only includes one line of handwritten text.

5. Conclusion

In this paper, we describe a method for separating lines of handwritten text in historical documents. We use preprocessing steps that remove uneven backgrounds, choose a reasonable threshold value, and remove solid lines (such as rule lines) from the image. We then create a binarized black/white transition count map (or alternatively, a binarized ALCM) to find probable locations of text lines. Text line components that appear to encompass more than one line of text are split using a min-cut/max-flow graph cut algorithm. Components that should probably be part of another nearby line are merged. Finally, a grayscale output image of each line is created, along with an associated foreground mask. Foreground components that belong to other lines of text are removed from both the grayscale image and the foreground mask. The foreground mask includes information about pixels that are ambiguous, and therefore may need to be ignored by subsequent processes (such as recognizers) when handling the line image.

Results of preliminary testing with our system are encouraging, as it performs well on a variety of historical documents from several different authors. Our method works well even on some very difficult documents.

Still, there are cases that our system does not always handle well. In this paper, we also identify some of these shortcomings, that we hope to address in future work as our system matures.

6. Future Work

As mentioned, we continue to acquire representative test images and will continue our tests and comparisons with other methods. While we so far only compare with a traditional profile method, we desire also to perform comparisons with other methods, such as the complete ALCM method described in [16].

From early analysis, we find that many of our errors are caused by shortcomings in our preprocessing. Improving our line removal method (or choosing a better method from

the literature) will help improve our results. Also of particular interest is preprocessing that will reduce the amount of bleed-through remaining on the page. Methods such as [15, 17] may prove to provide cleaner images than our current methods.

Our method will also be made more reliable by improving the criteria used in deciding when to merge line components, and to which other line components they should be merged.

Using a combination of methods may also provide better robustness. For example, using both an ALCM and the transition count map to take advantage of the strengths of each may prove worthwhile. And in cases where it can be determined that the handwritten lines are relatively straight and well-separated, reverting to the simple, fast profile-based methods may be in order.

References

[1] T. Akiyama and N. Hagita. Automated entry system for printed documents. *Pattern Recognition*, 23(11):1141–1154, 1990.

[2] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in computer vision. In *Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, September 2001.

[3] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26(9), 2004.

[4] L. A. D. Hutchison and W. A. Barrett. Fast registration of tabular document images using the fourier-mellin transform. In *IEEE First International Workshop on Document Image Analysis for Libraries*, pages 253–267, Palo Alto, California, January 2004.

[5] L. A. D. Hutchison and W. A. Barrett. Fourier-mellin registration of line-delineated tabular document images. *International Journal on Document Analysis and Recognition*, *(To Appear)*.

[6] E. Kavallieratou, N. Dromazou, N. Fakotakis, and G. Kokkinakis. An integrated system for handwritten document image processing. *International Journal of Pattern Recognition and Artificial Intelligence*, 17(4):617–636, 2003.

[7] V. Lavrenko, T. M. Rath, and R. Manmatha. Holistic word recognition for handwritten historical documents. In *Proceedings of the International Workshop on Document Image Analysis for Libraries (DIAL)*, pages 278–287, Palo Alto, CA, January 23-24, 2004.

[8] U. Mahadevan and R. C. Nagabushnam. Gap metrics for word separation in handwritten lines. In *International Conference on Document Analysis and Recognition, ICDAR*, pages 124–127, 1995.

[9] R. Manmatha and J. L. Rothfeder. A scale space approach for automatically segmenting words from historical handwritten documents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1212–1225, August 2005.

[10] W. Niblack. *An Introduction to Digital Image Processing*, pages 115–116. Prentice Hall, Englewood Cliffs, N.J., 1986.

[11] S. Nicolas, T. Paquet, and L. Heutte. Text line segmentation in handwritten document using a production system. In *9th IAPR International Workshop on Frontiers in Handwriting Recognition, IWFHR-9*, pages 245–250, Tokyo, Japan, 2004.

[12] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-9(1):62–66, January 1979.

[13] G. Seni and E. Cohen. External word segmentation of off-line handwritten text lines. *Pattern Recognition*, 27(1):41–52, 1994.

[14] A. W. Senior and A. J. Robinson. An off-line cursive handwriting recognition system. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):309–321, March 1998.

[15] Z. Shi and V. Govindaraju. Historical handwritten document image segmentation using background light intensity normalization. In *SPIE Document Recognition and Retrieval XII*, San Jose, California, 16-20 January 2005.

[16] Z. Shi, S. Setlur, and V. Govindaraju. Text extraction from gray scale historical document images using adaptive local connectivity map. In *8th International Conference on Document Analysis and Recognition, ICDAR*, volume 2, pages 794–798, Seoul, Korea, August 2005.

[17] Q. Wang, T. Xia, C. L. Tan, and L. Li. Image enhancement of historical documents using directional wavelet. *International Journal of Wavelets, Multiresolution and Information Processing*, 1(3):291–305, September 2003.

[18] B. A. Yanikoglu and P. A. Sandon. Segmentation of off-line cursive handwriting using linear programming. *Pattern Recognition*, 31(12):1825–1833, 1998.