International Journal of Document Analysis (2007) 10(1):39–52
DOI 10.1007/s10032-006-0019-8

ORIGINAL PAPER

E. Micah Kornfield · R. Manmatha · James Allan

Further explorations in text alignment with handwritten documents

Received: 21 December 2004 / Revised: 24 July 2005 / Accepted: 15 March 2006 / Published online: 11 May 2006

Abstract Today's digital libraries increasingly include not only printed text but also scanned handwritten pages and other multimedia material. There are, however, few tools available for manipulating handwritten pages. Here, we extend our algorithm from [5] based on dynamic time warping (DTW) for a word by word alignment of handwritten documents with (ASCII) transcripts. We specifically attempt to incorporate language modelling and parameter training into our algorithm. In addition, we take a critical look at our evaluation metrics. We see at least three uses for such alignment algorithms. First, alignment algorithms allow us to produce displays (for example on the web) that allow a person to easily find their place in the manuscript when reading a transcript. Second, such alignment algorithms will allow us to produce large quantities of ground truth data for evaluating handwriting recognition algorithms. Third, such algorithms allow us to produce indices in a straightforward manner for handwriting material. We provide experimental results of our algorithm on a set of 100 pages of historical handwritten material–specifically the writings of George Washington. Our method achieves average F-measure values of 68.3 on line by line alignment and 57.8 accuracy when aligning whole pages at time.

Keywords Aligning handwriting and transcript · Dynamic Time Warping

1 Introduction

A number of today's digital libraries contain handwritten material. Some of these libraries include both handwritten material and ASCII transcripts. An example of such a digital library is the Newton Project (http://www.newtonproject.ic.ac.uk) that proposes to create ASCII transcripts for Newton's handwritten manuscripts. Such historical manuscripts are hard to read. A word by word alignment of the transcript and the handwritten manuscript would allow a person to easily read the manuscript. It would also allow him or her to find their place in the manuscript using the transcript. For example, one could display both the manuscript and the transcript and whenever the mouse is held over a word in the transcript, the corresponding word in the manuscript would be outlined using a box.

Such alignments have other applications. One such application is the ability to create ground truth data for evaluating handwriting recognition and retrieval algorithms [11]. Effectively producing ground truth data for large collections of handwritten manuscripts is a manually intensive and laborious process that requires a person to first create a transcript based on the entire manuscript and then label individual words. The process of labelling can be avoided if alignment algorithms are available. Alignment also allows us to create an index for the manuscript. Specifically, this allows one to search the manuscript by searching its ASCII transcript. The alignment can then be used to highlight the search terms in the manuscript (as is done with conventional text search engines).

Creating such alignments is challenging since the transcript is an ASCII document while the manuscript page is an image. Handwriting recognition is not accurate enough to recognize such large vocabulary historical document collections. We therefore propose an alternative approach to aligning such material. The handwritten page image is automatically segmented. Features (for example box and text position, aspect ratio etc) are then computed for both the transcript and the page image. An algorithm based on dynamic time warping (DTW) is then used to align the words on the page image and the transcript. We compute alignments for whole pages and also for situations in which one can assume that the beginning and end positions of lines are known. We show results on a set of 100 pages from George Washington's handwriting.

Alignment is difficult because every step in the above mentioned approach produces errors. Segmentation of hand-

E. M. Kornfield · R. Manmatha · J. Allan (✉)
Center for Intelligent Information Retrieval, Department of Computer Science, University of Massachusetts Amherst, MA, USA
E-mail: {emkorn, manmatha, allan}@cs.umass.edu

writing is known to cause errors–both over and under segmentation occur. Since our corpus consists of scanned images of old historical documents, there are even more errors. In addition, the alignment algorithm itself produces errors.

Our prior work [5] consisted of using a DTW model for performing alignment. It assumed that images must be aligned with ASCII text from the transcript. In this paper our research is extended in several ways. First, we take a closer look at the estimation of our DTW features. Second, we introduce language modelling into our framework by adapting methods described in [4]. Third, we train values for feature weights and local path costs. Fourth, we examine the possibility of using a more complex local continuity constraint. Fifth, we introduce one more evaluation measure and provide empirical analysis of our proposed evaluation measures. Finally, we report results on a larger number of documents (100 vs. 70).

The remainder of this paper is organized as follows. Section 2 discusses related work and how our approach differs. We then continue by formally defining the problem and notation used for the rest of the paper in Sect. 3. In Sect. 4 we discuss the format of our data. Several baseline algorithms are discussed in Sect. 5. Section 6 goes over different evaluation metrics for the alignment tasks. Our DTW algorithm is described in Sect. 7. We conclude with experimental results in Sect. 8 and our conclusions along with a discussion of future research paths in Sect. 9.

2 Previous work

2.1 Historical documents

Very little research has been done on aligning transcripts of historical documents. As far as we know few people have examined the problem of aligning transcripts with handwritten documents.

Tomai et al. [14] investigated aligning transcripts with handwritten documents. The method they propose is to limit the lexicon of a handwriting recognizer by using the transcript. A ranked list of possible words from the lexicon is returned for each recognized word image. Several different likely segmentations of a line are made. The segmentation that has the highest probability given the transcript and previous alignments is then used. If a mapping cannot be performed with high enough confidence for a word then it is left out.

Tomai et al. give a figure of 82.95% accuracy in mapping words to a page. However, this figure makes certain assumptions. First they exclude 32 of the 249 words due to their "extreme noisiness". Including all words, their accuracy is roughly 72%. Second, they mention that of the 180 words they map, 17 are exactly mapped and 163 'roughly mapped'. In the absence of other information, we are unable to decide what the term 'roughly mapped' means and will assume that all 180 words were accurately mapped from transcript to manuscript. Finally, the results are reported for a single page of handwriting.

There has also been work in the areas of Automatic Speech Recognition (ASR) [12] and machine translation [4] on alignment. We note that these problems are somewhat different. For example, in machine translation, the alignment is between ASCII text in two different languages and additional constraints in terms of dictionary and grammar are available that are not available for word images.

2.2 Optical character recognition (OCR)

The document recognition community [2] has done research into aligning transcripts with machine printed documents for the purposes of creating ground truth.

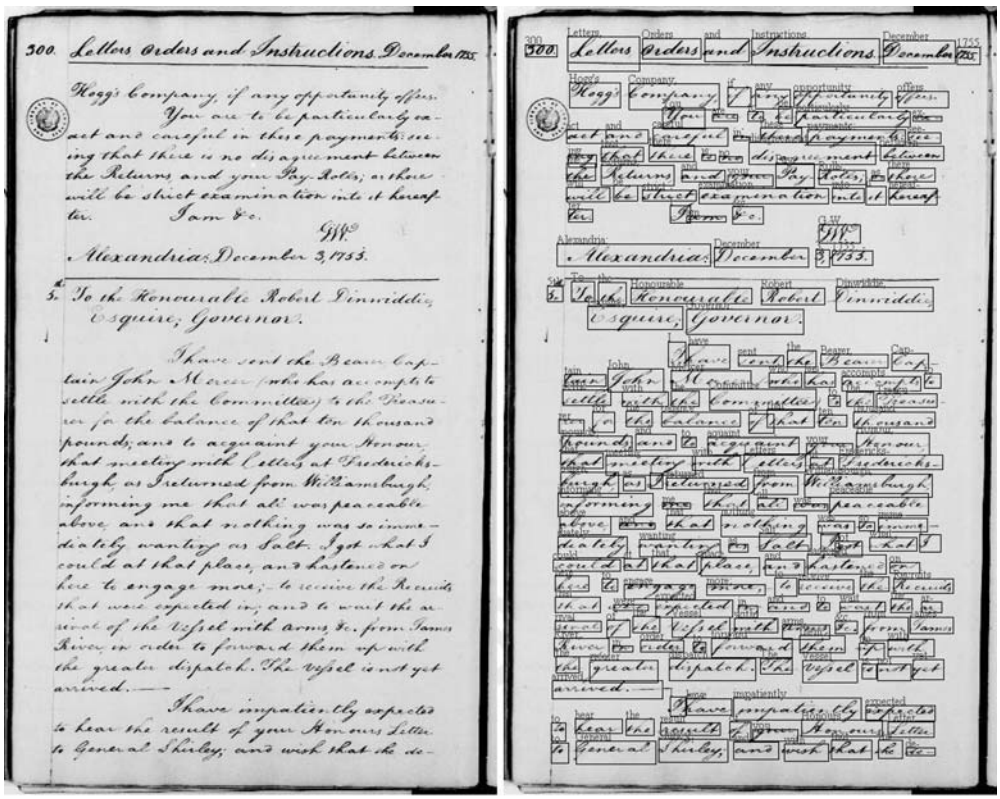
For example [2] tries to find a geometric transformation between the document description and the image of the document which minimizes a cost function. This technique assumes that along with the transcript there is a page description that denotes where the words in the transcript appear on the page. The most information that might be available in existing transcripts of historical documents is where line breaks occur. This limited information does not appear to be sufficient to make use of the algorithm proposed.

Another technique was proposed by Ho and Nagy [1]. Their proposed algorithm uses a predefined lexicon to help recognize characters. Ho and Nagy's algorithm is to segment a printed page into individual characters and cluster each of the segments. After clustering, character labels are assigned to the clusters by finding mappings that maximize a v/p ratio. The v/p ratio measures how well a set of mappings matches the lexicon. This technique is not directly applicable to our task because in general segmenting individual characters from handwritten manuscripts is very difficult. However, the idea of using the word-level language model from the transcripts to make assignments is appealing and similar to that of [4] which we use in this paper.

3 Problem definition and notation

Given a digitized image of a page D_i (the set of all pages is denoted by D) we generate a segmentation $\beta(D_i)$ that produces a vector of word images $\{b_0, b_1, \ldots, b_M\}$. For clarity, a segmentation actually produces bounding boxes for a digitized image, the pixels within a bounding box comprise a word image. We also have a transcript T_i that is a vector of ASCII words $\{w_0, w_1, \ldots, w_N\}$ for each page. For each $b_m \in \beta(D_i)$ we wish to select a set W_m of words from the transcript ($W_m \subseteq T_i \cup \{\}$) such that W_m contains the ASCII equivalent to what is represented by the word image b_m. An example of a handwritten page and a perfect alignment for the page is shown in Fig. 1.

When performing alignment we can view a segmented document $\beta(D_i)$ as containing multiple lines. Transcripts, however, might not contain such line breaks. In general,



(a) Handwritten Page (b) Perfect Alignment

Fig. 1 Handwritten page and perfect alignment

when we refer to $\beta(D_i)$, we view the entire document as one long line. This is accomplished by placing each successive line at the end of the previous line. For example, if we have two lines $\{b_1, \ldots, b_n\}$ and $\{b_{n+1}, \ldots, b_m\}$. We adjust every bounding box in $\{b_{n+1}, \ldots, b_m\}$ to have the same baseline (y-coordinate) as the first line ($\{b_1, \ldots, b_n\}$) and adjust the starting x-coordinate of each box in the second line by adding the x-coordinate of the end of image b_n.

Sometimes transcripts will have line break information. In this case, it is useful to remove the abstraction of a single long line and refer to specific lines. We denote this as $\lambda_l(\beta(D_i))$ where l indicates that we are interested in only the bounding boxes on the lth line. Similarly $\lambda_l(T_i)$ denotes we are interested only in the ASCII words on the corresponding line l of the transcript. $|\lambda(x)|$ gives the count of lines in either transcript or segmentation data.

4 Data

Our data consisted of 100 digitized pages from George Washington's archive. For each page we have two different types of segmentations with annotations and a line aligned transcript.

Table 1 Number of bounding boxes and lines in our evaluation data

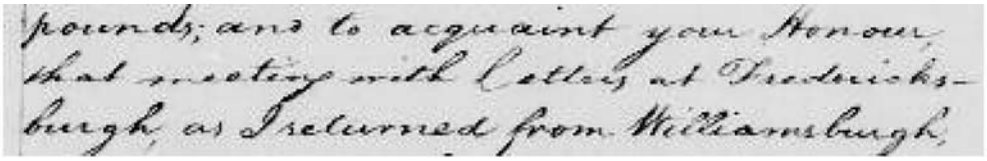
Segmentation	Number of boxes	Number of lines
Automatic (β^{auto})	25,213	3,379
Manual (β^{hand})	24,671	3,425

4.1 Segmentation (β)

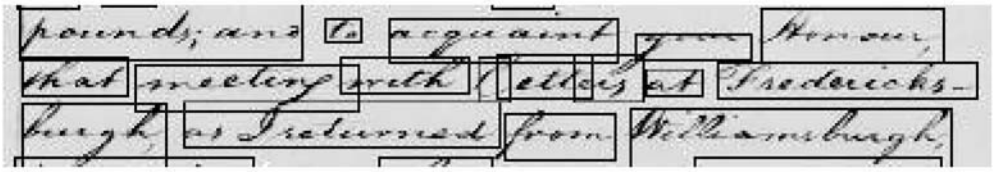
The segmentation produces a list of bounding boxes that when applied to the image should isolate all the pixels that are part of a single word. For each bounding box we have the coordinates that defines a rectangle and an indicator of the line in the digital image the bounding box occurs on. The two different types of segmentation are described below. Figure 2 shows each type of segmentation. Table 1 contains the number of boxes and lines in the segmentations for the 100 pages.

Automatic segmentations (β^{auto}) Automatic segmentations are those generated automatically by a program that is an improved version of [8] described in [7]. These segmentations are not perfect and can contain four different types of mistakes:

1. Bounding boxes will sometimes be placed around artifacts on the page that are not real words.



(a) 3 Lines from a sample image



(b) Automatic Segmentation

(c) Manual Segmentation

Fig. 2 An example of automatic and manual segmentation

2. Some words might have no bounding boxes placed around them.
3. Bounding boxes are sometimes placed around more then one word (under segmentation).
4. A word can sometimes be split into more then one bounding box (over segmentation), or only be partially included in a bounding box.

Manual segmentations (β^{hand}) Manual segmentations are corrections of automatically segmented pages. For each page an annotator corrected the automatic segmentations to create a one-to-one and onto mapping of words from the transcript to bounding boxes. Words in this case are strings made from all alphanumeric characters. It is important to note that with manual segmentations, alignment is trivial. We only use these segmentations for validating different aspects of our system. That is, the manual segmentations are considered ground truth.

4.2 Annotations ($A[\beta(D)]$)

Annotations consist of vectors of ASCII strings for each bounding box in a segmentation. These labels provide us with the true value of the contents of each bounding box, that can be used to evaluate how well or poorly an alignment algorithm works.

For manually segmented documents an annotation is simply the ASCII text equivalent of the word in the bounding box. Automatically segmented pages have a slightly richer representation to account for possible errors in the segmentation. For each bounding box that contains one or more words, the string labels are the exact text that is located within the bounding box (if a bounding box only covers part of a word, only the part covered is included). If a bounding box only contains part of a word, then in addition to exactly what is contained inside the box, we also record the complete word that was split by the box.

4.3 Transcripts (T)

A transcript is an ASCII text file consisting of text that corresponds to a specific page. Each file is aligned in parallel, on the line level, with the two different segmentations above. A transcript for a document is the same thing as an annotation for a hand segmented document image with some additional punctuation. It contains an exact match for the text in the document image.

Figure 3 contains an example transcript for the three lines contained in Fig. 2.

```
pounds; and to acquaint your Honour,

that meeting with Letters at Fredericks-

burgh, as I returned from Williamsburgh,
```

Fig. 3 Example transcript

5 Baseline algorithms

Baseline algorithms are fairly simple, naive algorithms that give us a reference point for determining how well our algorithm performs.

5.1 Linear alignment

Linear alignment is the simplest possible type of alignment one can imagine. If we have a set of bounding boxes $\{b_1, \ldots, b_M\}$ and a set of transcript words $\{w_1, \ldots, w_N\}$ we can do a forward alignment by assigning w_i to b_i where $1 \leq i \leq \min(M, N)$. Alternatively, we can start from the end of the document and move to the beginning by assigning w_{N-i-1} to b_{M-i-1} where $1 \leq i \leq \min(M, N)$. Note that when $N \neq M$, these techniques leave some words or bounding boxes unassigned.

5.2 Alignment using character position

Alignment using character position is done by aligning words and bounding boxes by calculating a normalized character position for either boxes or words and then finding the closest word to the position in the other set. In contrast to linear alignment, we are now trying to align words and boxes based on their length, rather than counting from the beginning or end. We define

- $\boldsymbol{X_{\text{start}}(b)}$ The starting x-coordinate of the bounding box for word image b.
- $\boldsymbol{X_{\text{end}}(b)}$ The ending x-coordinate of the bounding box for word image b.
- $\boldsymbol{Y_{\text{top}}(b)}$, $\boldsymbol{Y_{\text{bottom}}(b)}$ The corresponding quantities for the y-coordinate.
- $\boldsymbol{\mu}(\{\boldsymbol{b_i}, \ldots, \boldsymbol{b_{i+n}}\})$ is the width of a set of images and is equal to $X_{\text{end}}(b_{i+n}) - X_{\text{start}}(b_i)$. Note that if $n = 0$ then this definition is simply the width of the word image b_i. In addition, by defining width in this way, for any value of $n \geq 1$ spaces between word images are included in the width. The rationale behind this method of calculating width is that it will still provide accurate position estimates even if a word fails to be segmented as long as a mistake was not made at the beginning or end of a line.

The alignment can work in one of two ways, either from text to images or from images to text.

When aligning from text to images we calculate for each $w_1, \ldots, w_N$ the character position ($\text{CP}(w_j)$) as:

$$\text{CP}(w_j) = \frac{\sum_{i=1}^{j}(|w_i| + 1)}{\mu(\{w_1, \ldots, w_N\})} \tag{1}$$

We then multiply $\text{CP}(w_j)$ by $\mu(\{b_1, \ldots, b_M\})$. The resulting product, p_j, is a position somewhere in the interval $0 \leq p_j \leq \mu(b_1, \ldots, b_M)$. Box b_l is assigned to word w_j if position p_j lies somewhere within the box i.e. such that $X_{\text{start}}(b_l) \leq p_j \leq X_{\text{end}}(b_l)$. If p_j falls between two bounding boxes, then it is assigned to the closest of the two boxes b_{l+1}, b_l by computing $(\arg_b \min(X_{\text{start}}(b_{l+1}) - p_j, p_j - X_{\text{end}}(b_l)))$.

Alignment can also be performed by calculating the estimated character position in the images and multiplying it by the character width to get the position. The ratio is calculated as:

$$\text{CP}(b_j) = \frac{X_{\text{end}}(b_j)}{\mu(\{b_1, \ldots, b_M\})} \tag{2}$$

We then multiply by the width of the transcript ($\mu(\{w_1, \ldots, w_N\})$). The resulting product is a character position. If the character happens to be the space we arbitrarily pick the word preceding the space as the one to assign to box b_j.

5.3 Upper bound alignment

In upper bound alignment we try to assign the correct word with the correct box. Note that if a bounding box encircles two words, this alignment causes both ASCII words to be assigned to this box. This measure allows us to see what the maximum value of an evaluation metric we can expect, without performing the more complicated task of splitting ASCII words. It is generated automatically by assigning the complete word annotations to each box (see Sect. 4.2).

6 Alignment evaluation

Evaluation of the alignment is not straightforward. Evaluation metrics vary depending upon the goal of the alignment. For instance, if we are interested in generating training data for other handwriting recognition or retrieval algorithms, then we wish to have exact annotations for each bounding box. Alternatively, to build an index directly from alignments and use it for retrieval, a less strict measure might give a better idea of the results we can expect when conducting retrieval. Described below are four different evaluation methods that we use to evaluate alignments we generate.

Our evaluation measures are all defined by giving a score, on a bounding box level and then averaging this score for all of the bounding boxes in all of the documents.

6.1 Exact matching (σ_{exact})

For $b_j \in \beta(D_i)$ of a document we have an annotation, $S_j \in A[\beta(D_i)]$, and an alignment vector W_j (the words assigned to b_j). Exact matching gives a point (1) for b_j if $|S_j| = |W_j|$ and $\forall i : \{1 \leq i \leq |S_j|\} s_i = w_i$. That is, the two strings

are equal if they are the same length and all corresponding characters are equal. So

$$\sigma_{\text{exact}}(b_j) = \begin{cases} 1 & |S_j| = |W_j|, \forall i : \{1 \leq i \leq |S_j|\}\ s_i = w_i \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Exact matching is very strict. For a perfect score, it requires algorithms to not only give a reasonable alignment, but to trim words from the transcript to fit poorly segmented words and split words if a segmentation splits the word. This type of measure is probably best used when evaluating alignments for use as training data for other retrieval methods.

6.2 Edit distance matching (σ_{ED})

Exact match is a rigorous evaluation measure, and might not be suited to all applications of the alignment algorithm. We therefore propose a more relaxed definition of what it means to get an alignment for a bounding box correct. If we concatenate the strings in both our annotation for a bounding box and the aligned text for the box we can then use the value returned by Eq. (4) for the two resulting strings to judge if a bounding box has the correct text in it.

$$\sigma_{\text{ED}}(s_1, s_2) = \begin{cases} 1 & \max(|s_1|, |s_2|)/2 > \text{ED}(s_1, s_2) \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

where $\text{ED}(s_1, s_2)$ is the edit (Levenshtein) distance [6] between the two strings (s_1 and s_2). The edit distance between two strings is given by the recurrence:

$$\begin{aligned} \text{ED}(\text{“”}, \text{“”}) &= 0 \\ \text{ED}(s, \text{“”}) &= \text{ED}(\text{“”}, s) = |s| \\ \text{ED}(s_1 + c_1, s_2 + c_2) &= \min \begin{pmatrix} \text{ED}(s_1, s_2) + \epsilon(c_1, c_2), \\ \text{ED}(s_1 + c_1, s_2) + 1, \\ \text{ED}(s_1, s_2 + c_2) + 1 \end{pmatrix} \end{aligned}$$

where c_1, c_2 are characters, s is a string and $\epsilon(c_1, c_2)$ returns zero if the characters are equal and 1 otherwise. Edit distance matching is more relaxed then exact matching. By counting bounding boxes as correct if the words mostly match (the edit distance is less than half of the maximum of the lengths of the strings which are compared), it better reflects the case of using alignments for direct retrieval. It also give a little bit of leeway in case of annotation and transcript discrepancies caused by typographical errors in the creation of either set. So if we define $\kappa(\{st_1, \ldots, st_n\})$ to be the concatenation of a set of strings then

$$\sigma_{\text{ED}}(b_j) = \sigma_{\text{ED}}(\kappa(S_j), \kappa(W_j)) \quad (5)$$

6.3 Precision-recall and F-measure ($\sigma_{\text{Precision}}, \sigma_{\text{Recall}}, \sigma_{F\text{-measure}}$)

Recall and precision are measure commonly used in the information retrieval domain. We can extend them to alignment evaluation by calculating each of the metrics on a bounding box level. Precision is then defined as:

$$\text{precision}(S_j, W_j) = \frac{|S_j \cap W_j|}{|W_j|} \quad (6)$$

(the proportion of the words in the assignment that match the annotation) and recall as:

$$\text{recall}(S_j, W_j) = \frac{|S_j \cap W_j|}{|S_j|} \quad (7)$$

the proportion of the words in the annotation that are matched. So $\sigma_{\text{Precision}}(b_j) = \text{precision}(S_j, W_j)$ and $\sigma_{\text{Recall}}(b_j) = \text{recall}(S_j, W_j)$.

The F-measure is another commonly used metric used for information retrieval. It was proposed to make comparison of systems easier by combining recall and precision into a single number. The general F-measure is defined as:

$$\sigma_{F\text{-measure}}(b_j) = \frac{1}{\alpha \frac{1}{\sigma_{\text{Precision}}(b_j)} + (1 - \alpha) \frac{1}{\sigma_{\text{Recall}}}}$$

where α is a constant that weights either precision and recall depending on their relative importance in the evaluation. In our case we use the standard setting of $\alpha = 0.5$ so that:

$$\sigma_{F\text{-measure}}(b_j) = \frac{2\sigma_{\text{Precision}}(b_j)\sigma_{\text{Recall}}(b_j)}{\sigma_{\text{Precision}}(b_j) + \sigma_{\text{Recall}}}$$

6.4 Tomai et al. evaluation

The evaluation metric that is used by Tomai et al. [14], is slightly different in flavor then any of our proposed evaluation metrics. Instead of looking at bounding boxes and determining which words are placed correctly within a given box, they look at each transcript word and determine if the box it is mapped to contains the correct image. More formally for each word-box pair (w_i, b_j^{auto}), the mapping is considered correct if $w_i = S_k \in A[\beta^{\text{hand}}(D_i)]$ and

$$\begin{aligned} Y_{\text{top}}(b_j^{\text{auto}}) &\leq Y_{\text{top}}(b_k^{\text{hand}}) \\ Y_{\text{bottom}}(b_j^{\text{auto}}) &\geq Y_{\text{bottom}}(b_k^{\text{hand}}) \\ X_{\text{start}}(b_j^{\text{auto}}) &\leq X_{\text{start}}(b_k^{\text{hand}}) \\ X_{\text{end}}(b_j^{\text{auto}}) &\geq X_{\text{end}}(b_k^{\text{hand}}) \end{aligned}$$

Their score is calculated as the number of correct mappings divided by the size of the transcript. After careful consideration we believe that Tomai et. al's evaluation method

does not provide a good metric for how we have defined our task. Specifically, the constraints of the evaluation metric that ensure boxes are placed correctly directly conflicts with our notion of trying to determine which segments contain multiple words. When we integrate our alignment and segmentations systems (see Section 9) then this metric would be directly applicable to give us more a more complete evaluation of the new system.

6.5 Averaging

For any of the measures above, we can average the evaluation in three different ways. The first is over documents: (10).

$$\frac{\sum_{x=1}^{|D|}\left(\frac{\sum_{i=1}^{|\beta(D_x)|}\sigma(b_i)}{|\beta(D_x)|}\right)}{|D|} \tag{8}$$

That is, each page D_i is weighted equally. Recall that D is the set of handwritten document, D_i is a page, $\lambda_l(\beta(D_i))$ is a line and b_i is a word image. We can also weight each line equally:

$$\frac{\sum_{x=1}^{|D|}\sum_{i=1}^{|\lambda(D_x)|}(\sum_{b_y\in\lambda_i(\beta_i(D_x))}\sigma(b_y))}{\sum_{x=1}^{|D|}|\lambda(D_x)|} \tag{9}$$

or each word image equally:

$$\frac{\sum_{x=1}^{|D|}\sum_{i=1}^{|\beta(D_x)|}\sigma(b_i)}{\sum_{x=1}^{|D|}|\beta(D_x)|} \tag{10}$$

6.6 Stop words and evaluation

Evaluation depends upon the end-goal for our algorithm. We must decide which word alignments are important to us. For instance, when doing retrieval it is not important to match stop words correctly (because most retrieval systems remove them from a query in a preprocessing step). On the other hand, when creating ground truth data, it is more desirable to do well on all words in the document. We therefore analyze not only the overall system performance, but the performance after removing stop words from the evaluation as well.

7 Dynamic time warping

Dynamic Time Warping (DTW) is an algorithm for aligning two time series by minimizing the "distance" between them. A time series is a list of samples taken from a signal, ordered by the time that the respective samples were obtained. For our alignment task, we view each ASCII word in a transcript and each box in a segmentation as the samples that make up the two time series we are concerned with.

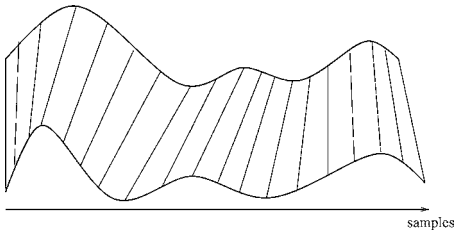


Fig. 4 Two similar time series aligned via Dynamic Time Warping. The lines between the two time series depict the assignment of corresponding points between the two time series

Rather than mapping samples that have the same time index to each other, DTW allows for the fact that one time signal may be warped with respect to the other. An example of an alignment for two series can be seen in Fig. 4. The name Time Warping is derived from the fact that this alignment "warps" the time axes of the two series so that the corresponding samples more closely relate to our intuition of what a good alignment should be. Intuitively what this means is for each possible assignment of some w_i to b_j we try to determine whether we should move forward in one or both of the time series to make an optimum assignment (one that minimizes cost) between subsequent sample points. The actual set of positions we can move to in one step of the algorithm is known as the local continuity constraint. In [5] we assumed that at each point we could either move forward a single step in both the word images ($\beta(D_i)$) and in transcripts (T_i), or one of them individually. This required that no word or word-image would be left out of the alignment. In this work, we expand the local continuity constraint to allow for moves of both one and two in any direction (see Fig. 5 for a graphical depiction of this constraint). Intuitively, this new constraint relaxes the original constraint by allowing the algorithm to skip a box, word, or both in the assignment process. This has the ability to aid in alignment by possibly detecting if a word was never segmented or if a word image contains garbage. Such occurrences are rare.

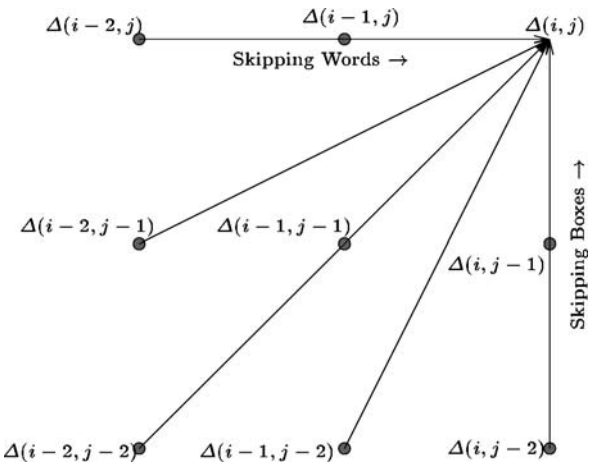


Fig. 5 A graphical depiction of the new local continuity constraint for DTW (Δ). To find the minimum cost path for word-image i and a transcript word j the algorithm examines previous points along the path that are a maximum of two units away along either axis

For example, less than 1% of the words in a page are missed for the word segmentation algorithm used here [7]. Consecutive occurrences–which would require two skips–are even rarer and hence are not considered.

Let the DTW cost between two time series $b_1 \ldots b_M$ and $w_1 \ldots w_N$ be $\mathrm{DTW}(M, N)$. $\mathrm{DTW}(M, N)$ is calculated using the following recurrence relation:

$$\mathrm{DTW}(i, j) = \min \left\{ \begin{array}{l} \mathrm{DTW}(i, j-1) + \rho_1 \\ \mathrm{DTW}(i-1, j) + \rho_2 \\ \mathrm{DTW}(i-1, j-1) + \rho_3 \\ \mathrm{DTW}(i, j-2) + \rho_4 \\ \mathrm{DTW}(i-2, j) + \rho_5 \\ \mathrm{DTW}(i-2, j-2) + \rho_6 \\ \mathrm{DTW}(i-1, j-2) + \rho_7 \\ \mathrm{DTW}(i-2, j-1) + \rho_8 \end{array} \right\} + d(b_i, w_j) \qquad (11)$$

where $d(b_i, w_j)$ is our sample-wise cost measure:

$$d(b_i, w_j) = \sum_{k=1}^{|\delta|} \eta_k * \delta_k(b_i, w_j) \qquad (12)$$

$\delta_k(b, w)$ is the kth word-box cost feature used (see Sect. 7.2) and η_k is a weight for the feature. The ρ's are costs associated with moving in the given direction of the warp. The directional costs (ρ_x) are present to protect against the DTW algorithm skipping as many points as possible to minimize cost. Additionally, the directional costs can be useful in both the original and new continuity constraints, by biasing the algorithm to move in a given direction. Both the ηs and ρs are determined by training (see Sect. 7.1).

7.1 Training

We used the Downhill-Simplex Algorithm ([9]) for training all the weights in our system (η and ρ). Downhill-Simplex is essentially a form of hill-climbing. It seemed well suited to our task for two reasons. First, it does not require explicit knowledge of the gradient. Second, it converges relatively quickly when compared with other learning techniques such as genetic algorithms. We used the F-measure (see Sect. 6) evaluation technique as an objective function for Downhill-Simplex.

An important aspect of DTW is that we constrain how much each of the time axes can be warped. This has a twofold effect. First, it reduces computation time for the algorithm. Second, it disallows large warps. By a large warp, we mean either assigning a single word to a large number of boxes, or a large number of words to a single box. This constraint is known as a global path constraint. There are a variety of ways that the global path constraint can be implemented. We chose to use the Sakoe–Chiba [13] band constraint that simply limits how far off the diagonal an alignment can move (see Fig. 6). The algorithm must satisfy both the global path constraint and the local continuity constraints. So for example, positions which satisfy the local continuity constraints will be eliminated from consideration if they lie outside the Sakoe–Chiba band.

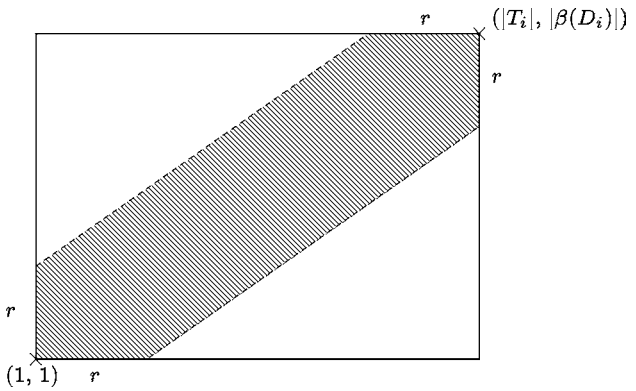


Fig. 6 Sakoe–Chiba path constraint with width r on the dynamic programming table. (1, 1) indicates the first point evaluated in the dynamic program. ($|T_i|$, $|\beta(D_i)|$) is the final point evaluated in the dynamic program (i.e. the end of both the text transcript and the segmentation). Only points within the shaded regions are evaluated in the dynamic program

Pseudocode for the algorithm is given in Fig. 7. Assignments are made by back tracking through the dynamic programming table starting at point ($|T_i|$,$|\beta(D_i)|$) and finding the preceding minimum point as defined by the recurrence.

7.2 Word-box features

Word-box features are used in calculating the cost of assigning a word to a given bounding box in DTW. Any combination of the features listed below can be used when running dynamic time warping. We used two distinct types of features. The first type relies on computing scalar features over the word images and ASCII text. Once we have feature values corresponding to each word in the transcript and image in the segmentation, we can then calculate the cost of any word-box pair using a suitable cost measure. In this case δ_k from Eq. (12) is defined as $\mathrm{cost}(f_k(w_i), f_k(b_j))$ where f_k is a feature below and cost is a cost function. There are many possible cost functions that can be used. [5] determined that in general an absolute difference ($\mathrm{cost}(x, y) = |x - y|$) works best.

Aspect ratio For an image b we calculate the aspect ratio as $\frac{Y_{\mathrm{bottom}}(b) - Y_{\mathrm{top}}(b)}{\mu(b)}$. There are two possible ways to calculate aspect ratio for text. The first is by rendering the text in a script font and performing the computation on the bounding box of the rendered word. The second is to take the height of a word to be constant and divide by the number of characters in the word.

Width For a word image width is calculated as $\frac{\mu(b_j)}{\sum_{b \in \beta(D)} \mu(b)}$. Similar to aspect ratio, there are two methods for calculating the width of ASCII words. The first is by rendering all the words and performing the computation on the rendered text images. The second is to use character count as the width, and perform the normalization based upon the total number of characters in the transcript.

Input: $T_i = (w_1, \ldots, w_{|T_i|})$ and $\beta(D_i) = (b_1, \ldots, b_{|\beta(D_i)|})$, cost function $d(\cdot, \cdot)$

Output: DTW matrix Δ

Algorithm:

1. $\Delta(1,1) = d(w_1, b_1)$;
2. for $m = 1{:}|T_i|$
3. $\quad \Delta(m,1) = \Delta(m-1,1) + d(w_m, b_1)$;
4. for $n = 1 : |\beta(D_i)|$
5. $\quad \Delta(1,n) = \Delta(1,n-1) + d(w_1, b_n)$;
6. for $m = 1 : |T_i|$
7. $\quad$ for $n = 1 : |\beta(D_i)|$
8. $$\Delta(m,n) = \min \left\{ \begin{array}{l} \Delta(m,n-1) + \rho_1 \\ \Delta(m-1,n) + \rho_2 \\ \Delta(m-1,n-1) + \rho_3 \\ \Delta(m,n-2) + \rho_4 \\ \Delta(m-2,n) + \rho_5 \\ \Delta(m-2,n-2) + \rho_6 \\ \Delta(m-1,n-2) + \rho_7 \\ \Delta(m-2,n-1) + \rho_8 \end{array} \right\} + d(w_m, b_n);$$

Fig. 7 Pseudocode for DTW (adapted from [15])

Character position We use Eqs. (1) and (2) to compute character positions. An alternative to calculating ASCII character position is to render all the words and use the analogue of Eq. (2) on the rendered words.

Ascender count Some characters have ascenders that extend above other characters. For instance capital letters, "l" and "d" have ascenders. An estimation technique [3] is used to try to determine the number of ascenders for a word image. Characters with ascenders can be directly counted for words from the transcript. All values are normalized to be between 0 and 1 with a mean of 0.5.

Descender count Some letters have descenders that extend below the baseline. For instance, "g" and "y" have descenders. The same techniques for finding ascenders is used for finding descenders in images and words. All values are normalized to be between 0 and 1 with a mean of 0.5.

The second type of cost feature does not explicitly extract two scalar values that can be compared with a simple cost function. Instead the cost for assigning a given word to an image is more complex. Two features we looked at were:

Stop word matching Stop word matching (SWM) gives a fixed penalty if we believe a word image contains a stop word ("a", "the", etc.) and the corresponding ASCII word is not aligned with the image. We target stop words because a relatively small number occur with a high frequency through out English documents. Our belief of the contents of a word image is based on trained clustering of all word images offline. More specifically we have a set of labeled clusters C such that $c \in C$ has a label representing the words in the cluster (i.e. "the", "a", etc). c is composed of a set of word images. $\delta_S(w_i, b_j)$ is defined as follows: if $\exists c \in C$ such that $b_j \in c$ then if $w_i \neq \text{label}(c)$ add a fixed penalty. Otherwise add zero. Clustering for stop word matching was done as follows:

1. Randomly arrange all word images we wish to cluster.
2. Using training data, build a cluster for each of the words we are interested in recognizing. We choose the most frequently occurring words that comprise 50% of the corpus. This is feasible due to the Zipfian distribution of documents in the English language.
3. Take the next image, b_i, to calculate its distance from each cluster: Find $\min_{c \in C}(\text{dist}(b_i, \psi(c)))$ and $\arg_c \min_{c \in C}(\text{dist}(b_i, \psi(c)))$. Where dist is the DTW distance [10] between the centroid of the cluster, $\psi(c)$, and the image.
4. If the distance in step 3 is less then a threshold γ (obtained through experimentation) then assign the image b_i to cluster c and update the centroid. Otherwise discard the example.
5. If there are more images to cluster go to step 3.

Word co-occurrence model We attempt to model the co-occurrences between word images and transcript words as an additional feature for our DTW algorithm. We adapt the algorithm proposed by Kay and Röscheisen [4] for our task. The original algorithm is meant for aligning sentences in a pair of parallel corpora of text in two different languages. To adapt the algorithm for our purposes we first need a vocabulary of visterms (a labeling of word images that allows us to refer to similar word images by the same label) to describe the word images. Creating visterms was accomplished by using untrained clustering. The untrained clustering algorithm is the same as that used for SWM with the exception that if the word image is not added to an existing cluster, a new cluster is added to the pool of clusters with its centroid set to the value of the word-image. Our visterm vocabulary then consists of cluster labels (each label is an arbitrarily assigned number for each cluster).

A second consideration when adapting the algorithm for our needs was how to determine sentence boundaries. A priori we have no knowledge of sentence boundaries

in $\beta(D_i)$ With transcripts we might have the necessary punctuation but it is possible that some documents may have been transcribed without punctuation. Another possibility might be to consider a line as a sentence, but in general as mentioned above transcripts might not have line boundary information. To solve this problem we say that every sentence is simply a unigram from either the transcript (a single word) or the document image (a single segment).

The algorithm [4] is an iterative process where in each iteration we make a hypothesis about which words from the parallel corpora correspond to one another. We then use these assignments to narrow down the choice of other possible assignments in the next iteration. The algorithm works as follows:

1. Enumerate all of the possible assignments of words to word images subject to a skew constraint and fixed points.

   The skew constraint limits the range of possible assignments by making the assumption that words and their corresponding word images are less then a certain distance away. Specifically the constraint enforces that for a possible assignment of a word w_i to a word image b_j the following must hold $|j-i| \leq \mu$. μ is the possible skew between the two corpora. The constraint is analogous to the global path constraint in DTW.

   Fixed points include page boundaries and correspondences between words and word images determined in an earlier iteration of the algorithm. Fixed points add an additional limitation on possible assignments. The limiting is achieved by further pruning possible assignments subject to the constraint that an assignment does not cross a fixed point. For example, consider a fixed point (x, y) (where x and y are indices of a word and visterm respectively) there are no possible assignments w_i, b_j where $i \leq x$ and $j > y$ or vice versa.
2. For each word-visterm pair find the likelihood that the word and visterm correspond. Afterwards prune unlikely word-visterm pairs and create a list sorted by descending correspondence likelihood.

   For every word-visterm pair, (l_m, v_n), where l_m is a word in the lexicon of T and v_n is a visterm created by clustering, calculate the statistical similarity of the pair. Similarity is calculated by:

$$\omega(l_m, v_n) = \frac{2\xi}{N_T(l_m) + N_\beta(v_n)} \tag{13}$$

   where ξ is a count of the number of times the word and its visterm might co-occur throughout the entire corpora (as determined by the possibilities that were enumerated in Step 1), and $N_J(x)$ is the frequency of x in corpus J.

   Eliminate all word-visterm pairs which are below a threshold. In order to find high-quality correspondences we use a threshold of τ standard deviations from the mean similarity of all word pairs, where τ is an adjustable parameter of the algorithm.

   In addition to the thresholding we impose the requirement that all word pairs (l_m, v_n) satisfy the following condition: $N_\beta(v_n) > 1, N_T(l_m) > 1$. This constraint eliminates singleton pairs. This is necessary because every singleton word-visterm pair has a similarity (ω) of 1.0, so by including them one is almost guaranteed to generate spurious matches.

   After calculating and culling word-visterm pairs we group them together by the frequency of words ($N_T(l_m)$). In our implementation we had three groups, one for $N_T(l_m) \geq 35$, one for $35 > N_T(l_m) \geq 10$ and one for $10 > N_T(l_m)$ The numbers are estimated by looking at a Zipfian distribution of the words in the transcript. 35 is close to the knee. The 10 is much lower down and is a conservative estimate since some of the corresponding boxes in the word image may be broken up. Within each group word-visterm pairs are sorted by descending similarity. Intuitively the different groups are a mechanism for providing different levels of confidence in the similarity score. For words with high frequencies we are more confident that our estimates of their similarities are accurate. In contrast, words with lower frequencies are more likely to co-occur as a random event. Therefore, the final step in making the list is to simply order the groups from highest to lowest frequency and save the result.
3. Extract possible assignments from the list created in Step 2.

   Read through the list sequentially, for each word-visterm pair identify the possible assignments from Step 1 that correspond to the pair. Specifically for a word-visterm pair (l_m, v_n) find all assignments (w_i, b_j) such that $w_i = l_m$ and v_n is the visterm for b_j. If all the assignments do not conflict (conflicting means that an assignment crosses a fixed point as defined in step 1) with any previous assignments, keep the assignments for (l_m, v_n) .
4. Make all the assignments kept in Step 3 fixed points.
5. Iterate through steps 1 through 4 until no new entries are added as alignments in Step 3. Once this occurs lower τ. For our implementation we use $1.0 \leq \tau \leq 2.0$.

After running this algorithm, we have a list of word-visterm pairs. We then assign penalties using the same method from Stop Word Matching for each word image that is described by a visterm in the list.

8 Experiments and results

Our experiments consisted of performing alignments for each transcript with both β^{auto} and β^{hand}. For each combination of transcript and segmented image we tested line-by-line (using line break information) alignment as well as

Table 2 A comparison of evaluation metrics illustrated via several baselines

	Upper bound	Linear alignment (from front)	Character position ($\beta \rightarrow T$)
Exact match	81.6	55.0	45.3
Recall	81.9	54.3	45.8
Precision	81.9	57.8	48.4
F-measure	81.9	55.4	46.4
Edit distance	87.9	64	48.9

Table 3 A comparison of rendered and unrendered features using the F-measure

	Rendered		Unrendered	
	Line by line	Page at a time	Line by line	Page at a time
Aspect ratio	56.0	35.1	45.9	8.8
Character position	62.4	6.8	61.9	6.7
Width	62	50.2	61.9	48.8

page at a time alignment (ignoring line break information). We first determined which method (character based or rendered) of computing aspect ratio, character position, and width provided better features for DTW alignment. We continued by comparing Stop Word Matching and the Word Co-occurrence Model as features. Afterwards, we attempted to train weights (η) for each feature in the cost function. Following the training of weights we looked at the effects of training directional path costs (ρ) in both the original continuity constraint (see [5]) and the extended constraint described in Sect. 7. For all experiments we used fivefold cross validation on 100 pages. We had separate training runs for cases where we used line break information and those in which we ignored line break information.

8.1 Metric comparison

Before discussing our experimental results an examination of our different evaluation methods is in order. When examining the range of values of the different metrics discussed in Sect. 6 we see some general trends. Experimental data confirms our intuition about exact match and edit distance metrics. Across the board the edit distance metric evaluates alignments with the highest percentage correct. Also, the exact match measurement tends, in general, to give the minimum score for an alignment. In addition, precision, recall and the F-measure techniques tend to be fairly close to one another and end up being someplace within the range of exact match and edit distance. For the remainder of the paper we report results using the F-measure. The choice of F-measure stems from two reasons. First, it is fair in the sense that it is a median of our algorithm's performance. It is neither the maximum or minimum measure in any of our experiments. Secondly, it encompasses both recall and precision giving a better idea of what can actually happen in the system. Table 2 shows the different results that occur when applying different metrics to three baseline alignment types. Out of the different averaging methods we chose to use box level averaging. The motivation for using this type of averaging is ultimately we care most about how well we can align the words with the boxes.

8.2 Rendered vs. unrendered features

In order to test which methods for calculating DTW features, rendered or unrendered, were superior we tried running DTW using only a single feature. For each type of feature we tried both an unrendered and rendered version. The results of these runs are summarized in Table 3. It is clear that rendering text and calculating features based on the rendering performs equal to or better than using the simpler character based computations. We believe the difference is particularly pronounced in the case of aspect ratio due to ascenders and descenders affecting the height component of the measurement significantly.

8.3 Stop word matching vs. co-occurrence model

To evaluate the impact our two features that depend on clustering we ran DTW using aspect ratio, width and character position as features combined with either stop word matching or co-occurrence features. We also performed alignment runs using both stop word matching or co-occurrence features and using neither, to evaluate how complementary the two features are, and how much impact each provides.

As we can see from Table 4 stop word matching improves performance by a higher degree (49.1 vs. 48.8). But there is some benefit to using both models (an additional 0.4 of accuracy). We believe the overall impact of both features is low due to poor clustering performance. Intuitively the poor clustering also explains why stop word matching performs better. Without accurate clustering the co-occurrence algorithm would have a more difficult time finding likely matches. However, inaccurate clustering would simply cause some spurious identification of stop words, which can be corrected by the other features included in DTW.

Table 4 A comparison of stop word matching and the co-occurrence model as features using the F-measure

	Line by line	Page at a time
Neither stop word or co-occurrence	67.1	48.2
Co-ocurrence	67.1	48.8
Stop word	67.2	49.1
Both	67.2	49.5

Table 5 F-Measure evaluation of basic alignment algorithms on aligning transcripts with automatically segmented pages

	Normal		Non stop words only	
	Line by line	Page at a time	Line by line	Page at a time
Linear alignment (from front)	46.6	3.6	39.1	1.6
Linear alignment (from back)	43.1	7.1	33.3	1.9
Character position ($T \rightarrow \beta^{\text{auto}}$)	52.0	7.9	48.3	5.9
Character position ($\beta^{\text{auto}} \rightarrow T$)	55.4	8.1	52.4	9.1
Upper bound	81.9	81.9	67.6	67.6

Table 6 Results (F-measure) of training on DTW

	Normal		Non stop words only	
	Line by line	Page at a time	Line by line	Page at a time
No training	67.2	55.4	57.7	46.9
Feature weight training	67.2	56.3	57.5	47.5
Feature and path training	67.2	57.8	57.4	48.7
Feature and extended path training	68.3	55.8	56.4	48.1

8.4 ASCII (T) to automatic segmentation (β^{auto})

The results of aligning transcripts with an automatically segmented page using the base-line algorithms described in Sect. 5 are presented in Table 5. These results are similar with those presented in [5].

In [5] we noted that when using line break information the character position feature helps performance but if we do not have the information character position becomes a hindrance. Keeping this result in mind for the rest of our experiments we include all features discussed in Sect. 7.2, including character position when doing line by line alignment. When doing alignment on pages without line break information the same features are used except that we omit character position. Where applicable we apply results from Sect. 8.2 and use rendered estimation for features.

Table 6 summarizes the results of training. Using normal evaluation (evaluating all words) the results indicate that training only helps the performance of the line by line alignment when we use the extended continuity constraint. Contrastingly, for alignment without line break information training we see performance gains using training, but the smallest gain is realized when using the extended path. Whether we used line break information or not, the performance on non-stop word alignment seems to be independent of increases in overall system performance.

We also wished to determine to what extent our clustering performance affects our system performance. To determine this we eliminated cluster performance issues by using perfect clustering (based on box labels). Table 7 shows the results of retraining weights using features based upon the perfect clustering. The results seem to indicate that when we have line break information, an increase in clustering performance will help only a small amount. However, when aligning entire documents at a time we see a much larger increase. Intuitively, this makes sense because with line by line alignment we have break points which allow us to restart the alignment from scratch. However, when aligning a page at a time stop word matching and the co-occurrence model serve as pseudo-breakpoints with which that algorithm can in some sense restart itself from scratch.

Table 7 Results (F-measure) of training on DTW using perfect clustering for stop word matching and co-occurrence model

	Normal		Non stop words only	
	Line by line	Page at a time	Line by line	Page at a time
No training	68.3	63.2	58.4	53.0
Feature weight training	70.0	65.5	59.7	54.7
Feature and path training	70.2	66.0	59.9	55.0
Feature and extended path training	70.0	65.9	59.4	54.9

Table 8 Results (F-measure) of basic alignment algorithms on aligning transcripts with hand segmented pages

	Normal		Non stop words only	
	Line by line	Page at a time	Line by line	Page at a time
Linear alignment (from front)	100.0	100.0	100.0	100.0
Linear alignment (from back)	100.0	100.0	100.0	100.0
Character position ($T \rightarrow \beta^{\text{auto}}$)	69.8	7.7	79.0	10.2
Character position ($\beta^{\text{auto}} \rightarrow T$)	84.8	17.9	92.3	23.0
Upper bound	100.0	100.0	100.0	100.0

Table 9 Results (F-measure) of training on DTW with hand segmented pages

	Normal		Non Stop Words Only	
	Line by line	Page at a time	Line by line	Page at a time
No training	99.8	93.8	99.8	93.6
Feature weight training	99.8	95.3	99.8	94.8
Feature and path training	99.8	98.9	99.8	98.6
Feature and extended path training	99.8	98.5	99.7	98.1

8.5 ASCII (T) to manual segmentation (β^{hand})

When aligning transcripts to hand-segmented pages (see Table 8) we did not retrain any parameters. If we had done so we would expect that training weights on the path constraints would have simply forced the algorithm to take the diagonal path on every occasion. DTW as before performs very well on this task (see Table 9). Training enables page at a time alignment to achieve an F-measure of 98.9 accuracy.

9 Conclusion and future work

Our DTW algorithm still outperforms any of the baseline measures by a fair margin. Training helps increase this margin slightly more in the case of page at a time alignment. But it seems that we need to augment the model to get further system performance. It is possible that a different local continuity constraint than the one presented in this paper might help. In addition, different machine learning algorithms might be able to find better feature and path weights. More investigation is needed into both of these possibilities.

Our results show that for the page at a time approach performance increases significantly with improvements in clustering performance. Further investigations into clustering or other methods for recognizing very common words will help improve our results further. In addition, it would be helpful to start investigation into methods for splitting words between boxes.

Ultimately, we still foresee the segmentation and alignment system working as an iterative process where each iteration refines the output, until no changes occur.

Further areas of research exist in trying to leverage imperfect transcripts of documents. For instance, it might be more expedient to read historical documents out loud and have an automatic speech recognition (ASR) system produce an ASCII transcript. Of course, ASR is not perfect and will introduce errors in the transcript. Developing algorithms to deal with the noisiness from both transcripts and segmentations will be even more challenging than the problem addressed in this paper.

Another challenging task to be addressed in the area of alignment is non-standard documents. For instance, it is not clear that our techniques that assume documents consist of prose, will also adapt to mathematical formulas and diagrams.

Acknowledgements This work was supported in part by the Center for Intelligent Information Retrieval and in part by the National Science Foundation under grant number IIS-9909073. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsor.

References

1. Ho, T., Nagy, G.: OCR with no shape training. In: Proceedings of 15th ICPR, pp. 27–30. Barcelona (2000)
2. Hobby, J.D.: Matching document images with ground truth. Int. J. Doc Anal. Recognit. **1**(1), 52–61 (1998)
3. Kane, S., Lehman, A., Partridge, E.: Indexing George Washington's handwritten manuscripts. Technical Report MM-34, Center for Intelligent Information Retrieval. University of Massachusetts Amherst (2001)
4. Kay, M., Röscheisen, M.: Text-translation alignment. Comput. Linguist. **19**(1), 121–142 (1993)
5. Kornfield, E.M., Manmatha, R., Allan, J.: Text alignment with handwritten documents. In: Proceedings of DIAL, pp.195–211. Palo Alto, California (2004)
6. Levenshtein, V.I.: Binary codes capable of correcting spurious insertions and deletions of ones. Russian Problemy Peredachi Informatsii **1**, 12–25 (1965) (Original in Russian. English translation in Problems of Information Transmission **1**, 8–17 (1965))
7. Manmatha, R., Rothfeder, J.: A scale space approach for automatically segmenting words from historical handwritten documents. IEEE Trans. Pattern Anal. Mach. Intell. **27**(8), 1212–1225 (2005)
8. Manmatha, R., Srimal, N.: Scale space technique for word segmentation in handwritten documents. In: Scale-Space Theories in Computer Vision pp. 22–33 (1999)
9. Press, W., Teukolsky, S., Vetterling, W., Flannery, B.: Numerical Recipes in C. Cambridge University Press, Cambridge, UK (1993)

10. Rath, T., Manmatha, R.: Word image matching using dynamic time warping. In: Proceedings of CVPR-03, vol. 2, pp. 521–527. Madison, WI (2003)
11. Rath, T.M., Lavrenko, V., Manmatha, R.: A statistical approach to retrieving historical manuscript images without recognition. Technical Report MM-42, Center for Intelligent Information Retrieval, University of Massachusetts Amherst (2003)
12. Roy, D.K., Malamud, C.: Speaker identification based text to audio alignment for an audio retrieval system. In: Proceedings of ICASSP '97, pp. 1099–1102. Munich, Germany (1997)
13. Sakoe, H., Chiba, S.: Dynamic programming optimization for spoken work recognition. IEEE Trans. Acoust. Speeh Signal Process. **26**, 623–625 (1980)
14. Tomai, C., Zhang, B., Govindaraju, V.: Transcript mapping for historic handwritten document images. In: Proceedings of the 8th International Workshop on Frontiers in Handwriting Recognition, pp. 413–418. Niagara-on-the-Lake, ON (2002)
15. Triebel, R.: Automatische erkennung von handgeschriebenen worten mithilfe des level-building algorithmus. Master's thesis, Institut fur Informatik, alber-Ludwigs-Universtät Freiburg (1999) (in German)