

Unsupervised Font Reconstruction Based on Token Co-occurrence

Michael P. Cutter
cutter@iupr.com

Joost van Beusekom
j_vbeu@iupr.com

Faisal Shafait
faisal.shafait@dfki.de

Thomas M. Breuel
tmb@cs.uni-kl.de

Technical University of
Kaiserslautern, Germany

ABSTRACT

High quality conversions of scanned documents into PDF usually either rely on full OCR or token compression. This paper describes an approach intermediate between those two: it is based on token clustering, but additionally groups tokens into candidate fonts. Our approach has the potential of yielding OCR-like PDFs when the inputs are high quality and degrading to token based compression when the font analysis fails, while preserving full visual fidelity. Our approach is based on an unsupervised algorithm for grouping tokens into candidate fonts. The algorithm constructs a graph based on token proximity and derives token groups by partitioning this graph. In initial experiments on scanned 300 dpi pages containing multiple fonts, this technique reconstructs candidate fonts with 100% accuracy.

Categories and Subject Descriptors

I.7.5 [Document and Text Processing]: Document Capture—*document analysis*; H.3.7 [Information Storage and Retrieval]: [Digital Libraries Collection]

General Terms

Algorithms

Keywords

Token Compression, Font Reconstruction, Candidate Fonts, Token Co-occurrence Graph Partitioning

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DocEng2010, September 21–24, 2010, Manchester, United Kingdom.
Copyright 2010 ACM 978-1-4503-0231-9/10/09 ...\$10.00.

1. INTRODUCTION

Automatically creating high quality versions of printed documents has become increasingly important as more and more books and archives are scanned. The implication of having every printed text available at the click of a mouse in a searchable, reflowable format is a greatly increased accessibility to knowledge. Online archives desire a structured format such as PDF/A, due to the advantage of compression and reflowability. In order to save space, it is desirable to utilize compression algorithms that compresses text and images while taking into account the content they are compressing.

The goal of font reconstruction is to replace each letter found in a document image with a Bézier curve representing the outline of the letter. The advantages of using a vector outline are increased compression, scalability and improved readability. Additionally, once a font has been reconstructed, it is also available for creating new text, a capability that is important in some historical contexts.

The primary contribution of this paper is a novel method that partitions the tokens found in a document during token-based compression into candidate reconstructed fonts. Token partitioning is an important aspect of our font reconstruction because it selects tokens representing clusters of letters from the document image to become part of a candidate font. Using the candidate font from this process, it is possible to classify which font every letter belongs to, and then the page can be reconstructed by using the font class of the letter and the parameterized reconstructed glyphs. It will also be possible to make additions to the original document without the requirement of knowing the original font(s) used. Through token partitioning, a candidate font is formed that is essentially a set of extracted glyphs from a scanned document image that can then be reused.

In this paper, we show that by using our technique it is possible to completely describe the fonts found in a document image with automatically grouped tokens. The grouped tokens or candidate font, result from a sequence of steps including: Optical Character Recognition (OCR), token based clustering, a token co-occurrence or proximity graph, and, finally, the greedy graph segmentation algorithm that assigns tokens to a candidate font. The OCR assists in segmenting connected components into letters by using a language model. The binned-nearest-neighbor algorithm is designed

to make use of the OCR results, and the distance metric used is designed to avoid false font merging. The partitioning algorithm is motivated by the assumption that words are typically written in the same font, so if the same token co-occurs in different words, then all of the letters in the linked words have a high probability of being in the same font. The candidate fonts are evaluated based on if each token in that candidate font comes from the same original font. We find through our evaluation in Section 4, that our technique perfectly partitions tokens into candidate fonts. These candidate fonts are an important step towards our goal of creating a system that gives full, reflowable PDF/A for clean document images, but degrades gracefully to a token-compressed image for bad inputs.

There does not appear to be much research on font reconstruction from scanned, unlabeled documents. One commercial product that offers this capability is the Adobe Acrobat version 9 feature called Clear Scan. While no publication describes the algorithm Clear Scan uses, based on published descriptions available, it performs something similar to font reconstruction. Their blog [1] makes a claim that with Clear Scan you can decrease the file size while improving the readability and greatly decreasing the time it takes to print the file. The most relevant related work to font reconstruction is font recognition. Previously published research in font recognition, described in the related work section, has primarily addressed the problem of classifying a document containing known fonts. There has been work in unsupervised clustering of fonts, but, as further discussed in the related work section, their approach is different from ours because they use a global approach. We address the problem differently, by segmenting with the help of OCR, then partitioning the letters found in a document into a new candidate font, without considering any previously known fonts.

The rest of the paper is structured as follows. In Section 2, the background of research techniques used for font identification and recognition is discussed. In Section 3, the method is described and broken down into four main phases: the OCR phase, the binned-nearest-neighbor token clustering phase, the co-occurrence graph construction phase, and, finally, the graph partitioning into candidate fonts. In Section 4, the evaluation of the quality of a candidate font is explained, along with discussion of the motivation for several aspects of the partitioning algorithm. Lastly, Section 5 highlights the planned future work.

2. RELATED WORK

Optical Font Recognition (OFR) is a niche of (OCR) that has been approached with various methods [12, 17]. The related work in font identification discussed in this section can be split into either character by character or global analysis approaches. All of the related character by character methods can be considered supervised OFC, since these techniques function by using labeled training data. First, they apply a feature extraction method on known fonts. They then apply the same feature extraction on unknown characters to assign the font classes based on a nearest neighbor approach. In 2000, Nagy wrote a survey of work in document image analysis [13] with a section devoted to font classification and script detection.

Recent character by character methods, such as *fyfont* [12], compare features from each connected component against a

font database in order to classify the font class of each component. The method performs an eigenvalue calculation after edge based feature extraction. By using eigenvalues, the feature space is reduced and therefore an increased speed of the query response of the font database is possible. The authors set up a website [3] where one can upload an image and the font can be identified with the help of labeling each connected component. Their method achieved high recognition accuracy and over 99% of the time the correct font is one of the top five suggested fonts. Zrammdini et al. [17] use a multivariate Bayesian classifier on typographical features extracted on a global scale. The technique also uses a known font dataset of hundred of fonts. Since it is a global approach, it can only classify the document with one font. A word by word technique developed by Khoubyari et al. [10] classifies the dominant font in the document by comparing frequent words such as articles and prepositions (a,and,the,etc) to a database of these words in various common fonts. The technique assumes that a document is only authored in one font.

The techniques discussed above in the area of supervised font identification do have high font recognition rates, but they are only suitable when recognizing a font that has already been encountered and labeled. This motivates the development of a novel automatic font partitioning algorithm that does not require as a prerequisite that the font found in the page already exists in some database.

There has been work in unsupervised font recognition proposed by Zhu et al. [16] and Avilées-Cruz et al. [5] using global texture windows. The method by Avilées-Cruz et al. first performs preprocessing on the document image to ensure monospacing, and then performs feature extraction using various sized texture windows; finally, the Expectation Maximization algorithm clusters the windows together into font groups.

One way our method differs from previous OFR [17, 15], is that it uses OCR to assist OFR instead of the other way around. It also contributes to the field with a novel technique that solves a problem not addressed in the previous work [5, 3, 10]: our technique can classify multiple fonts within a document without using any known font dataset. The goal of our font partitioning is to be able to reuse reconstructed fonts in new documents in order to facilitate reflowable, searchable, and compressed text.

3. METHOD

Candidate font reconstruction in this paper is achieved through a pipeline of Optical Character Recognition (OCR), Token Clustering (TC), and, finally, a Greedy Graph Segmentation (GGS) algorithm.

The candidate font reconstruction technique is applied only after OCR. The first step as shown in Figure 2 [Part (b)] is to perform token based clustering which clusters visually similar letters, i.e. letters that are the same font and alphabetical letter class are assigned the same token ID. A token represents all the similar letters merged into a representative image along with a count of the total letters associated with the token. The next step as shown in Figure 2 [Part C & D)] treats each token as a node in a graph where links between nodes represent how each token co-occurs in words found in the scanned document. The final step is to group the tokens based on the links between them.

He was so badly dressed that even a man accustomed to shabbiness would have

(a) DejaVu Serif

"It's in the houses of spiteful old widows that one finds such cleanliness," Raskolnikov

(b) AlMateen

"All that's nonsense," he said hopefully, "and there is nothing in it all to worry

(c) Sazanami Gothic

Figure 1: These are small snippets of the scanned document image containing multiple fonts. The images were scanned on a flat bed scanner at 300 dpi.

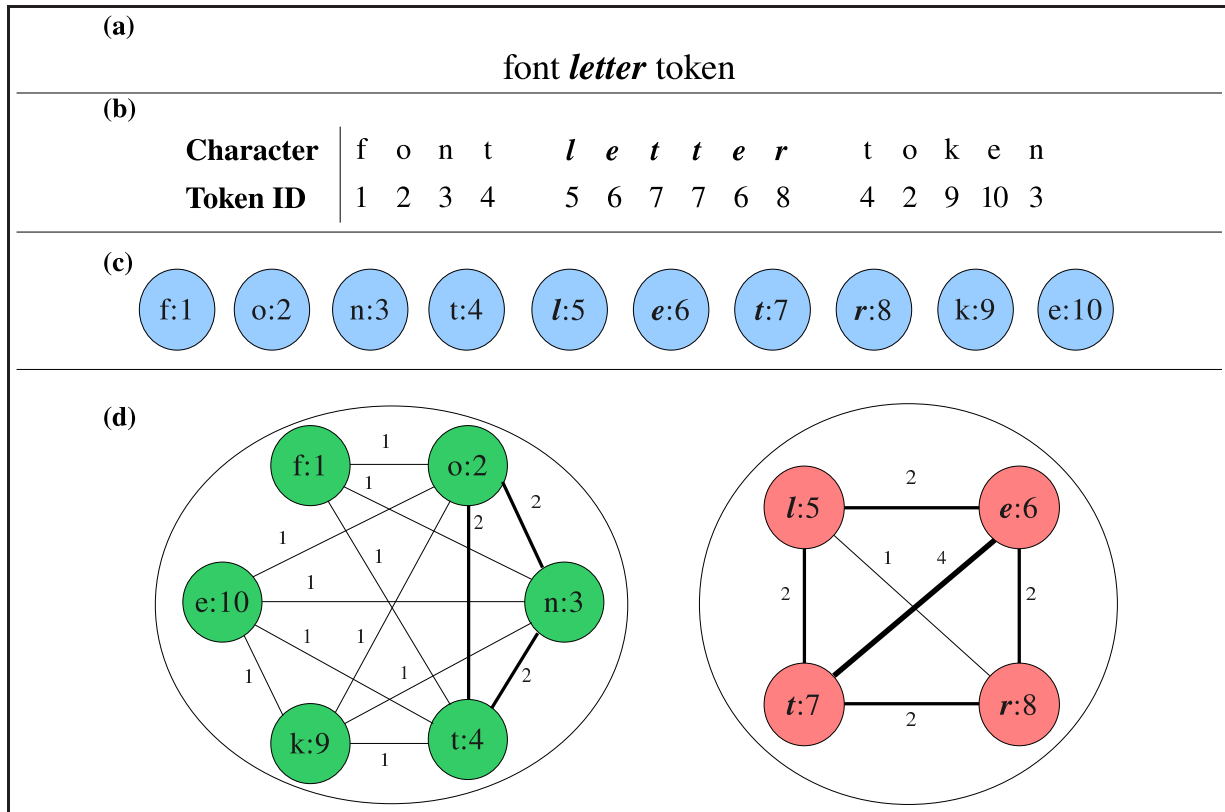


Figure 2: The candidate font reconstruction pipeline. (a) The input image. (b) Token clustering stage: letters are assigned token IDs. (c) Each token ID becomes a node. (d) Co-occurrence graph and candidate font partitioning: an edge weight represents every time there is a token co-occurrence with another token in the same word. Since token '4' and '3' co-occur in the word 'font' and in the word 'token', the edge between them has a weight value of two. Finally, a greedy graph segmentation algorithm groups tokens, connected by the token co-occurrence graph, into candidate fonts.



Figure 3: Example of how connected component extraction can fail. Here this image was recognized as containing three components 'R' 'a' and 'H'. The image is of the word 'Raft', but due to the font that it is typeset in and noise, the space between the 'f' and 't' is invisible. To a connected component extraction system, each horizontal white space delimited sequence of black pixels is considered a connected component.



Figure 4: Example of how using a statistical language model can yield better results. Here this image was recognized as 'Raft' because 'RaH' could not be found commonly in a corpus of English text.

3.1 OCR

Candidate font reconstruction technique is implemented after the traditional OCR pipeline has been performed by OCRopus [4, 6, 7, 8]. Fortunately, OCRopus, a state of the art OCR open source program, handles the difficult process of character segmentation, which allows our technique to begin with letters that have been segmented and classified. Our contribution, therefore, is focused on clustering these letters into tokens and then grouping these tokens into candidate reconstructed fonts.

OCR is an initial step in the pipeline for this method because it is essential for font reconstruction that character segmentation is performed as well as possible. Previous font identification methods have simply extracted connected components from the scanned document image, which can lead to problems, such as shown in Figure 3. In this image, connected component extraction recognized adjacent letters 'f' and 't' as just a strange 'H'; unfortunately, this kind of under-segmentation mistake is far too common. That is why the OCR engine used in our approach invokes a statistical language model to segment words into letters versus connected components. The statistical language model rejects the segmentation shown in Figure 3 because 'RaH' is not a word. It then finds the closest probabilistic segmentation that would render this image into a word, which results in a segmentation shown in Figure 4.

Using an OCR engine is also integral to our process because in order to form a candidate font because it is necessary to know the class label of each letter. Additionally, the candidate font must be accurately labeled in order to use the font to reprint the document. If an OCR system encounters a new font when it has not been trained for the specific nuances of the font by an appearance model, er-

ror rates sharply increase. The method is tolerant to OCR error because of the greedy approach of the partitioning algorithm. Therefore, even a high OCR error rate does not imply that the letters in the newly reconstructed candidate font are necessarily mislabeled. These labeled letters are the input to the next phase of the technique, binned nearest neighbor clustering.

3.2 Binned-Nearest-Neighbor Clustering

This candidate font reconstruction method relies on first grouping letters as seen in Figure 6 into tokens during the token clustering phase, see Figure 2 [part b]. In order to discover which letters together form a font, we first group letters into clusters. These clusters of letters are considered tokens and will later be partitioned from the token co-occurrence into various candidate fonts.

Token clustering is a common procedure that is analogous to the well-known symbol compression scheme used in the JBIG2 standard. Token clustering merges images that are visually similar together. For our candidate font reconstruction method, we have to take special precautions in order to successfully reconstruct a candidate font.

- i. The clustering algorithm must be efficient enough to run on a variety of resource-constrained platforms. A standard nearest neighbor model must compare each added letter to every already existing token. Applying this standard classifier to a book image, which may contain many thousands, if not hundreds of thousands, of letters, requires too many comparisons for an acceptable run time.
- ii. Two very similar letters with the same character class and only slight differences based on their font should not be merged into a token. If they are merged, then the reconstructed font will not be accurate because the token will have come from two different fonts.

In order to ensure a fast processing time, we modified the standard nearest neighbor model to include binning based on image features. Simple preprocessing is performed on an image of a letter so that it is only compared to other letters that could be potential matches for clustering. The preprocessing extracts features: height, width, and the number of holes present in the image. These features form a hash that determine which bin the token is restricted to.

The dissimilarity measure used to compute the nearest neighbor adds weight heavily to the differences in the outline of the images of letters; the motivation being, a candidate font should not contain two letters from different original fonts. This is achieved by first aligning the letters based on their centroids. A mask, as seen in Figure 5(b), of the image of the letter is computed by a combination of morphologic operations; the inverted, dilated image is subtracted from the binary eroded image and then a binary OR is computed with a thinned version of the image, as seen in Figure 5(c). This sequence of operations generates a mask that is used to compute the edge weighted dissimilarity metric, i.e. differences between two letters on the exterior of their shape results in a higher total error score. The computation of the error is computed on two images aligned by centroids by the following equation:

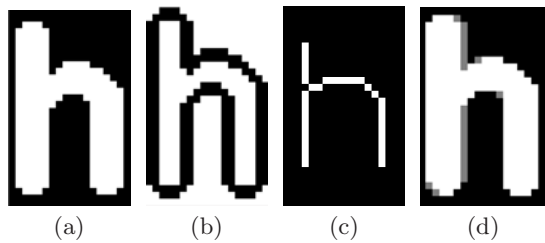


Figure 5: Various images of a 'h' as it goes through the binned-nearest-neighbor-clustering algorithm (a) Raw binarized inverted image of letter. (b) Mask of image - black pixels represent outline. (c) Thinned image - used to reduce weight on interior during distance computation. (d) Result of merging an image of a letter with a token prototype image. The new image pixels are averaged with the current token prototype with a weighted average depending on how many letters the token previously represented.

$$\text{error} = \sum_{x=0}^W \sum_{y=0}^H M(x,y) \times \|T(x,y) - I(x,y)\| \quad (1)$$

$$\text{error}_I = \sum_{x=0}^W \sum_{y=0}^H M(x,y) \times I(x,y) \quad (2)$$

$$\text{error}_T = \sum_{x=0}^W \sum_{y=0}^H M(x,y) \times T(x,y) \quad (3)$$

$$\text{FinalError} = \min\left(\frac{\text{error}}{\text{error}_I}, \frac{\text{error}}{\text{error}_T}\right) \quad (4)$$

Where I is the candidate image, T is the token, and M is the mask. H and W represent the height and width of the token respectively. The candidate image and token will always have same height and width as a prerequisite for being in the same bin.

A new image is compared against every other image in the same bin in order to find the match with the minimal dissimilarity score. If this dissimilarity score is below the threshold, the new image is merged with its strongest match, shown in Figure 5(d). On the other hand, if the dissimilarity score is above the threshold, then a new token cluster, that would look like Figure 5(a), is created for that image in its respective bin.

The benefit of this approach is to cluster letters together only if they are of the same font. Differences in fonts are generally seen on the exteriors of images such as serifs vs sans serifs. The serifs, the strokes found at the end of letters, distinguish two almost identical letters as a different font. Our clustering method is designed to be robust against this distinction.

This paper's binned-nearest-neighbor clustering algorithm's highlighted features are:

1. Clustering has the constraint that OCR class labels must match, which increases speed by decreasing the number of comparisons.



Figure 6: List of 'h' letters that the dissimilarity measure evaluates to be similar enough to belong to the same token – see Figure 5(d).

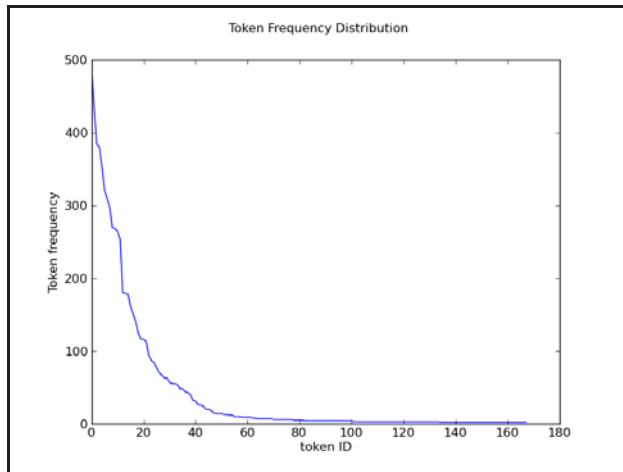


Figure 7: Token frequency distribution: the graph shows only the first 160 tokens. There are thousands of others but since they only represent one letter, they are omitted from this chart.

2. Binning criteria is affected by topology.
3. The outline of the image is weighted heavily to prevent false font merging.

3.3 Token Co-occurrence Graph

The unsupervised automatic font grouping method relies on the assumption that words are formed by letters of the same font. Our model tries to optimize the best grouping of token IDs based on their links between one another. A token co-occurrence graph is constructed as the feature inputted into the font grouping method.

The co-occurrence matrix measures how many times token IDs are found in the same word together. The co-occurrence is measured so that if tokens are found in many words together, then the edge is heavy that connects their respective node in the co-occurrence graph. The method used to segment the graph into candidate fonts is greedy and tries to find the maximum alphabet by finding the most strongly connected nodes.

3.4 Greedy Graph Segmentation

The goal of greedy graph segmentation is to approximate the fonts in a document by creating a candidate font for every font in the document. The edges between tokens represented in the token co-occurrence graph are the primary feature to partition the tokens into the candidate fonts. The algorithm's goal is to maximize the sum of all edge weights

between a token and every other token in a candidate font. In other words, the goal of greedy graph segmentation is to assign the optimal set of token IDs to candidate font F to maximize Equation 5 subject to constraints discussed below.

$$\text{LinkScore}(F_i) = \sum_{t_k \in F_i} \sum_{t_j \in F_i} G(t_k, t_j) \times H(j, k) \quad (5)$$

$$H(j, k) = \begin{cases} 1 & : j \neq k \\ 0 & : j = k \end{cases} \quad (6)$$

$$\forall t_k, t_j \in F_i \mid t_j \neq t_k \Rightarrow C(t_{id_k}) \neq C(t_{id_j}) \quad (7)$$

Where function $C(x)$ maps token ID x to its character class, $G(x, y)$ is the edge weight between node x and y , k and j are generic token IDs. Equation 7 represents the constraint that a candidate font can only contain one token for each character class in its alphabet. There is another constraint that a token ID can at most, only become a part of one candidate font. Additionally, a candidate font can be at most a set of 26 lowercase and 26 uppercase tokens, each of which represent clusters of alphabetical letters.

The Greedy Graph Segmentation Algorithm

1. The token co-occurrence graph is computed.
2. The token that represents the greatest number of letters and is not already in a candidate font becomes the seed for the new candidate font.
3. Tokens contained in candidate font’s edges are explored for nodes representing tokens that have a character class not in the current candidate font or are a member of any other candidate font, and in the case of a tie, the node with the most links to the current candidate font is added.
4. **Repeat** Step 3 until all character classes are found or until an iteration threshold has been reached.
5. Randomly choose a token from the candidate font and swap it with another token representing the same character class if the LinkScore is increased by switching to the new token and the new token is not in any other candidate font class and represents at least as many letters as the previous token.
6. **Repeat** Step 5 for the number of swaps that has been specified.
7. **Repeat** Step 2 for the number of target fonts that has been specified.

Greedy graph segmentation (GGS) is an algorithm that labels the nodes in the token co-occurrence graph into a candidate font. It is initialized by selecting the token that occurs most and using that token as the seed for a candidate font. Then greedy graph segmentation searches all of the surrounding edges of the tokens in a candidate font and adds new tokens to a list of potentials that are not already in the candidate font’s alphabet. After the search is complete, the potentials are added to the candidate font instantly, unless they have duplicate character classes. In this case, each potential is evaluated by a heuristic, with scoring based on

Table 1: Evaluation of GGS technique

	font 1	font 2	font 3
DejaVu Serif	100%	0	0
AlMateen	0	100%	0
Sazanami Gothic	0	0	100%

how strongly connected it is to the current candidate font. The potential with the higher score is assigned to the candidate font. Greedy graph segmentation runs iteratively until one of two criteria are met: the algorithm has run for over a certain threshold¹ of iterations, or every letter in the alphabet is represented in the candidate font. In practice, the former always occurs first because the probability of the document containing a capital "Z" and that was recognized by OCR correctly is extremely unlikely.

4. EVALUATION AND DISCUSSION

We applied this method to a ten page scanned document image that contained three fonts. The content for the document was generated using a digital copy of a historical text and then artificially changing each third of the document to a different font. The three fonts chosen for our test set were; DejaVu Serif (Figure 1(a)), AlMateen (Figure 1(b)), and Sazanami Gothic (Figure 1(c)). For this experiment, we applied our full technique just as in the simple example in Figure 2 except on a much larger document (snippets of this document can be seen in Figure 1).

Greedy graph segmentation was run with the parameter to search for three candidate fonts. The resulting candidate reconstructed fonts are evaluated based on only containing glyphs from each respective original font. Since this condition is met, we achieved 100% accuracy selecting tokens to become members of a font. In Figure 8, there is an example of the 'e' and 'f' from the original digital font and the candidate font.

Our evaluation of GGS is based on how well a candidate font represents the font it is approximating. The dataset was labeled so that each letter is given a ground-truth label as to which font it represents. The precision equation measures how closely the candidate font represents the actual font in the scanned document image. For each candidate font, an error ratio is calculated (Table 1). A true positive is when a token in a candidate font is selected from the correct ground-truth labeled font region. A false positive is when a token in a candidate font is part of a region of the page that is ground-truth labeled to be in another font. For our evaluation, we saw that tokens in each candidate font all came from the same font regions of the test images. This means our method was 100% accurate at partitioning tokens of different fonts into candidate fonts that approximated real fonts found in the scanned document image.

Often when applying tokenization techniques to a document, it is more important to not merge tokens together incorrectly than to leave them in separate clusters. In our approach, the threshold is set sufficiently high so if there is a potential chance of two letters being of slightly different fonts, they are put into separate candidate fonts. As seen in Figure 7, the motivation for the greedy aspect of

¹A value of 100 epochs has been shown to be sufficient; often after 10 epochs there are no new token IDs added to a candidate font




	Digital font	Reconstructed font
DejaVu Serif	ef	
AlMateen	ef	
Sazanami Gothic	ef	

Figure 8: This figure shows in the third column the current status of candidate fonts (generated by autotracing [2] bitmap outlines). It highlights the improvements necessary to recreate a document using font reconstruction.

the graph segmentation approach is evident. Letters that have been frequently merged together are excellent candidates for a candidate font because they have been smoothed by repeated mergers.

This algorithm makes the assumption that each word in the data it is examining is written in a single font. There are some rare situations where this assumption does not hold; some disciplines may mix different fonts within the same word e.g. "I used *MATLAB*'s *eig()* function to find the eigenvalues". An italic version of a font is considered a different font, both by our distance function and by definition. When analyzing a document containing mixed-font words, the token co-occurrence graph will contain a link between tokens that are not part of the same font. This could result in candidate fonts that contain glyphs from two different fonts. However, since the heuristic is greedy in respect to the number of links, it is very unlikely that in a large document a few mixed font words would cause candidate font confusion.

5. CONCLUSIONS AND FUTURE WORK

The research presented in this paper contributes a novel solution to the problem of partitioning tokens into candidate fonts. The final goal from our work in font reconstruction is the ability to digitally recreate scanned documents in a Mix Raster Content [14] (MRC) format. The first step in the process, font partitioning, is proposed. The result of font partitioning are groups of tokens for each font present in a document. As visible in Figure 8, the 'e' and 'f' are the same size, which needs to be corrected in order to reuse the partitioned glyphs to recreate the document.

The final step needed for full digital recreation will be to combine text image segmentation with font reconstruction. Some examples of other techniques that recreate documents using a MRC format are [11, 9]; it is planned for future work to benchmark our font reconstruction technique against these other similar methods.

6. ACKNOWLEDGMENTS

Parts of this project were financed by the Mellon Foundation's DECAPOD project

7. REFERENCES

- [1] Adobe's blog discussing Clear Scans advantages. http://blogs.adobe.com/acrolaw/2009/05/better_pdf_ocr_clearscan_is_small.html/.
- [2] AutoTrace - converts bitmap to vector graphics. autotrace.sourceforge.net/.
- [3] FyFont, a visual search engine for fonts. <http://media-vibrance.itn.liu.se/fyfont/>.
- [4] The OCRopus(tm) open source document analysis and OCR system. <http://code.google.com/p/ocropus/>.
- [5] C. Avilés-Cruz, J. Villegas, and R. Escarela-Perez. Unsupervised Font Clustering Using Stochastic Version of the EM Algorithm and Global Texture Analysis. In *Progress in Pattern Recognition, Image Analysis and Applications*, pages 3–25, 2004.
- [6] T. Breuel. The OCRopus Open Source OCR System. In B. Yanikoglou and K. Berkner, editors, *Proceedings of the Document and Retrieval XV, IS&T/SPIE 20th Annual Symposium 2008*. SPIE, 2008.
- [7] T. Breuel. Recent progress on the OCRopus OCR system. In *MOCR'09: Proceedings of the International Workshop on Multilingual OCR*, pages 1–10, New York, NY, USA, 2009. ACM.
- [8] T. M. Breuel. Segmentation of Handprinted Letter Strings Using a Dynamic Programming Algorithm. *Document Analysis and Recognition, International Conference on*, 0:0821, 2001.
- [9] P. Haffner, L. Bottou, P. G. Howard, and Y. Lecun. DjVu: Analyzing and Compressing Scanned Documents for Internet Distribution. In *In Proceedings of the International Conference on Document Analysis and Recognition*, pages 625–628, 1999.
- [10] S. Khoubiyari and J. J. Hull. Font and Function Word Identification in Document Recognition. *Computer Vision and Image Understanding*, 63(1):66–74, 1996.
- [11] A. Langley and D. S. Bloomberg. Google Books: Making the public domain universally accessible. In *Proceedings of SPIE Volume 6500, Document Recognition and Retrieval XIV*, pages 1–10, 2007.
- [12] R. L. Martin Solli. FyFont: Find-your-Font in Large Font Database. In *Image Analysis*, pages 432–441, 2007.
- [13] G. Nagy. Twenty Years of Document Image Analysis in PAMI. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:38–62, Jan. 2000.
- [14] R. D. Queiroz, R. Buckley, and M. Xu. Mixed raster content (MRC) model for compound image compression. In *Proc EI 99, VCIP, SPIE*, pages 1106–1117, 1999.
- [15] Y. Xu and G. Nagy. Prototype Extraction and Adaptive OCR. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21:1280–1296, 1999.
- [16] Y. Zhu, T. Tan, and Y. Wang. Font Recognition Based on Global Texture Analysis. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1192–1200, Oct. 2001.
- [17] A. Zramdini and R. Ingold. Optical Font Recognition Using Typographical Features. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, pages 877–882, August 1998.