Font Cluster Identification with Reconstructed Font Clusters

Michael P. Cutter, Joost van Beusekom, Faisal Shafait, Thomas M. Breuel

ABSTRACT

An accessible digital document should be searchable, compressed, highly readable and faithful to the original. These requirements can be achieved by digitally recreating the document with embedded fonts; however, it is not always known what fonts were used to author the original document. It is desirable to be able to reconstruct fonts with vector glyphs that approximate the shapes of characters that form a font. In this work we address the assignment of every character within the document to a font cluster, which is necessary to represent a scanned document image with a reconstructed font. This paper extends previous work in font reconstruction by proposing and evaluating an algorithm to assign a font to every character within a document. Through our evaluation method, the algorithm's font cluster assignment accuracy is measured to be 96% on multi-font documents.

Keywords: Font Reconstruction, Font Identification, Reconstructed Font, Token Matching

1. INTRODUCTION

Digital archives need to be sensitive to the needs of a global audience. In general, book capture and digitization increases the accessibility of literature in two ways. First, a digital book is not constrained to a physical location, instead it is available virtually to anyone with an Internet connection. Second, a digital book is more accessible to a global audience because each word can be recognized by Optical Character Recognition (OCR) recognized. The digital book's recognized words can be translated to another language and spoken aloud to a visually challenged or illiterate person.

Generally, in traditional OCR systems, when a digital character's class is recognized, the character's font is not. This creates a challenge for displaying a digital material faithfully to the original. Often the solution to this problem is to display an image of the document with the recognized text hidden underneath in a non-visible way. This allows a user to search the document while still seeing a visually faithful representation of the original document. However, because each word in a digital document with hidden text is glued to the word's original location, the document is constrained to a fixed format and aspect ratio. This problem motivates the solution of font reconstruction, which has the same advantages of searchability and visual accuracy without the same fixed format constraint. The original font can be reconstructed and then repurposed to regenerate the document faithfully. Using a digitally reconstructed font instead of hidden text also means that the document's glyphs are scalable and reflowable. Someone can read the book on a constrained resolution device, such as a mobile phone. Going the other direction, a visually challenged individual can blow the document up on a large monitor; either user sees a visually faithful version of the book regardless of the window size they view the book with.

The focus of this work is the extension of our previous work in font reconstruction.[1] The previous work's focus was to find a representative reconstructed font cluster for each font present in a document. Letters from the document are categorized as belonging in the same reconstructed font based on their co-location in the document's words. The reconstructed font cluster is meant to be a reconstructed font's alphabet consisting of letters that with a high degree of accuracy come from the same font. In previous published results, our technique worked perfectly. This paper's new contribution is an algorithm to assign all the remaining letters to a reconstructed font cluster. The algorithm utilizes both the letter's location on the page and shape to assign the letter to a font cluster. This assignment is necessary, because when the document is recreated with a reconstructed font, the proper font is used for each letter.

Font reconstruction has grown from efforts in font identification, which has been explored with supervised and unsupervised techniques. A similar technique was developed by Serdar et al.[2]to cluster fonts in an unsupervised fashion. They prepared a set of 4420 character images from 65 fonts, with bitmaps resized to 32x32. All of the characters' images' edges are filtered and then eigenimages are calculated. The final font cluster classification in their technique is addressed by randomly selecting characters from a region; the most common font among the selected characters is considered the dominate font of that region.

A method by Avilées-Cruz et al.[3] preprocesses the document image's words to make sure that the words are monospaced. Next, their technique performs feature extraction on various sized texture windows. Then their technique relies on the expectation maximization algorithm to cluster the texture windows into font groups.

Supervised font identification methods make use of large databases of fonts. Solli et al.[4,5] compare each character to a database of over two thousand fonts. Each character has many filters applied to it before the eigen His method ranks the likelihood of a character belonging to each font, and 99% of the time the correct font is one of the top five proposed. Khoubyaricviu et al.'s[6] font identification technique is similar, but instead of segmenting words into individual characters, their method compares articles such as 'the' to a font database.

The rest of the paper is structured as follows: in section 2 the previous automatic font clustering technique and new font identification algorithm are described, in section 3 our method is evaluated for different values of parameters, the results are discussed in section 4, and finally, in section 5 conclusions and future work are presented.

2. METHOD

In this paper, we extend the automatic font clustering technique for font reconstruction[1] by addressing multi-font document font classification using reconstructed font clusters. With these reconstructed font clusters, our goal is to assign the remaining letters represented by tokens to font clusters. This works contribution is a novel font identification algorithm that utilizes Equations 7 through 10 to classify the font cluster of the letters in a document.

Initially, a first pass font assignment algorithm attempts to group letters that are near other letters, which are already part of a reconstructed font cluster; by labeling the neigboring letters as part of the same font cluster. The first step is motivated by the assumption that almost always a word is written in the same font. Therefore Equation 7 groups tokens found near tokens that are part of reconstructed font clusters are assigned to that font cluster.

Then all original reconstructed font cluster's shapes become the training data for the reconstructed font's clusters' nearest neighbor model. Equation 10 is used to assign tokens that are not part of reconstructed font clusters to font clusters according to, which reconstructed font cluster has a token with the most similar shape.

2.1 Terminology

Letter: A letter is an image of a character segmented from its adjacent characters. In this work, the distinction between a character and a letter is that, a letter is segmented using a statistical language model using OCRopus[7–9] and, with a high degree of probability, only represents a single alphabetical letter.

Token: A token is the result of merging similar letters together. Each token has a unique ID, all of the original letters that are now represented by the token can be replaced by this token's ID.

Reconstructed Glyph: A reconstructed glyph is created by tracing the prototype token bitmap outline and adding necessary parameters so it can lay on the page the same it was found.

Token Co-occurrence Graph: The token co-occurrence graph is an abstraction of the layout of token IDs on a page of a document. Each node represents a token ID and each edge is formed when two token IDs are in the same uninterrupted sequence of token IDs.

Reconstructed Font: A reconstructed font is a collection of token IDs that have been clustered together. A reconstructed font represents a single alphabet including all the lower case and upper case characters. It is common however, for a reconstructed font to be missing several unlikly characters, such as a capital "Z".

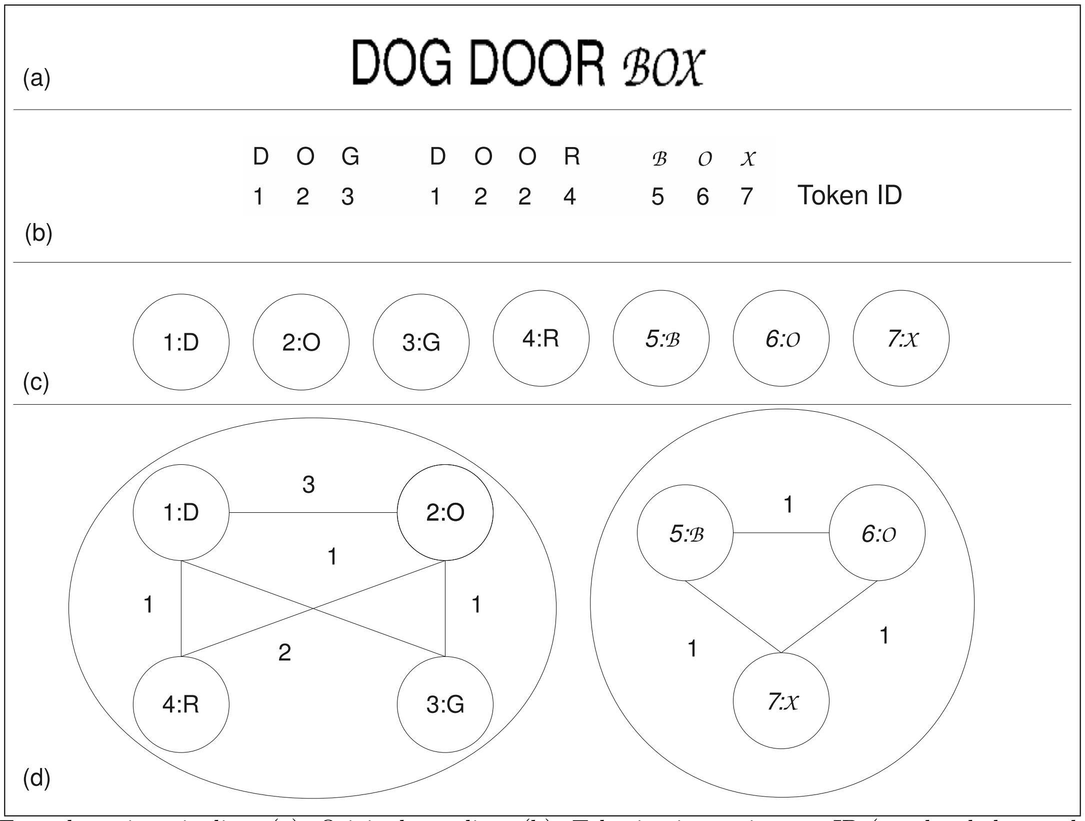


Figure 1. Font clustering pipeline. (a): Original text line. (b): Tokenization assigns an ID (number below each letter) to each connected component. (c): Each token ID is represented as a node. (d): Nodes are clustered into groups based on how they co-occur in words.

2.2 Automatic Font Clustering

Our previously published,[1] automatic font clustering method can be summarized graphically by Figure 1. The technique is explained in this section for the reader's ease in understanding the rest of the work.

The first step of the automatic font clustering technique is for the document to be recognized by OCR. The OCR system, OCRopus, outputs segmented labeled letters, which become the input to a font clustering algorithm. Letters' shapes with the same character label are compared using an edge sensitive weight metric, designed to only merge letters when they come from the same font and have the same character class.

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

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

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

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

In the previous equations, T is the token prototype and I is the image being assessed as a possible match. I and T are first aligned using their centroids. A mask, M, of the outline of the letter's shape is created, using mathematical morphology. For further details of the motivation and implementation of this clustering dissimilarity metric see.[1] Letters considered matches after this comparison are represented by the same token prototype, which is a blend of the letters' shapes it represents.

The document can now be represented as a sequence of token IDs and spaces (see Figure 1 part (b)). The next step is to construct a token co-occurrence graph, where the nodes are token IDs and each edge represents a co-occurrence of the token IDs in the same word. A word in this context is when a string of token IDs are interrupted by a recognized space. The token co-occurrence graph is segmented by a graph partitioning algorithm that finds the optimal set of token ids for each font cluster to have the highest possible LinkScore.

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

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

In Equation 5, F_i, is the font cluster and $G(x,y)$ is a function that gives the edge weight in the token co-occurrence graph between the node x and y. The graph partitioning algorithm is under the constraint that each font cluster, F_i, represents an alphabet, meaning that the same cluster can not have two tokens that represent 'a'. Additionally the optimization of LinkScore is under the constraint that a token can only belong to one font cluster or none at all.

The final output of the automatic font clustering technique are font clusters, F_i, which contain representative letters of a reconstructed fonts alphabet.

2.3 Font Cluster Indentification

The font identification algorithm described next, is the primary contribution of this paper. Both are applied after the entire automatic font clustering technique (See Figure 1), has been applied to the document. The goal of this font identification algorithm is to accurately label the font of all of the tokens that are not part of any reconstructed font cluster.

The first step is to apply a first pass proximity based labeling algorithm. In the following equations j is the index of a letter the reading order list of letters within a document, $f_{\hat{i}}$ is the assigned font class, f_i is a font class and R_{f_i} is the set of all character locations in a font cluster with the label f_i.

$$j_{f_{\hat{i}}} = \operatorname*{argmax}_{f_i \in f} g(f_i, j) \tag{7}$$

$$g(f_i, j) = \sum_{v=(j-K)}^{(j+K)} \frac{1}{\|j - v\|} \times h(v, f_i) \tag{8}$$

The algorithm uses Equation 7 for every j letters on the document, to determine the reconstructed font cluster to assign j to. Parameter K controls the reach of the first pass font labeling algorithm. The consequence for the decaying weight term(see Equation 8) is that a letter's font class assignment is weighted higher for the closest proximity neighboring font.

$$h(v, f_i) = \begin{cases} 1 & : x \in R_{f_i} \\ 0 & : x \notin R_{f_i} \end{cases} \tag{9}$$

The indicator Function 9 insures that only letters that are part of a font cluster can influence the font class assignment. If none of the k neighbors are part of a reconstructed font, the letter is given no label.

After the neighborhood proximity first pass labeling, the next step for the algorithm is to utilize the token's prototype shape to determine its reconstructed font class. Every token prototype in a document is resized to a length and width of 32 by 32. The two dimensional image is then converted to a one dimensional vector of length 1024 ($32 \times 32 = 1024$). The token's, that have been clustered together to form a reconstructed font, prototypes; become the training data for that reconstructed font cluster's (C_{f_i}) nearest neighbor model .

Every token that is not already in a reconstructed font is compared against these classifiers. The token's font class is determined by (Equation 10), taking the class of the font-classifier with the minimum euclidean distance (Equation 12); unless the distance is below the error threshold α (Equation 11).

$$f_i = \underset{I_j \in C_i \in C}{\operatorname{argmin}} \, d(I_j, I_\star) \tag{10}$$

$$d(I_j, I_\star) < \alpha \tag{11}$$

$$d(I_j, I_\star) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \tag{12}$$

Where C is the set of all classifiers, C_i, is a specific classifier for a font class, f_i is a font class, I is a letter image vector, I_j is equal to $x_1 + \cdots + x_n$ and $I_\star$ is equal to $y_1 + \cdots + y_n$.

3. EVALUATION

As an initial experiment of the font identification technique, we experimented on a ten page document image. The document was synthetically created to consist of three fonts. The advantage of synthetically creating the document is knowing for certain the fonts involved. The digital document is exported as an image and then fed as the input to the automatic font clustering pipeline (see Figure 1). Then the font identification algorithm discussed in section 2.3 is used to classify the font of each letter in the document.

We ran the experiment for different values of parameters α and K. Parameter α was tested with values between 0 and 5000 in intervals of 100. For each of these value of α, the parameter K was set successively from zero to four in intervals of one. With 50 values of α and 5 values of K, the algorithm was run a total of 250 times. The results of this sequence of experiments can be seen graphically in Figure 2 and Figure 3. We discuss the quality of our algorithm using the following metrics.

Coverage: defined by the number of letters in the document whose distance, governed by Equation 12, are less than the current level of α.

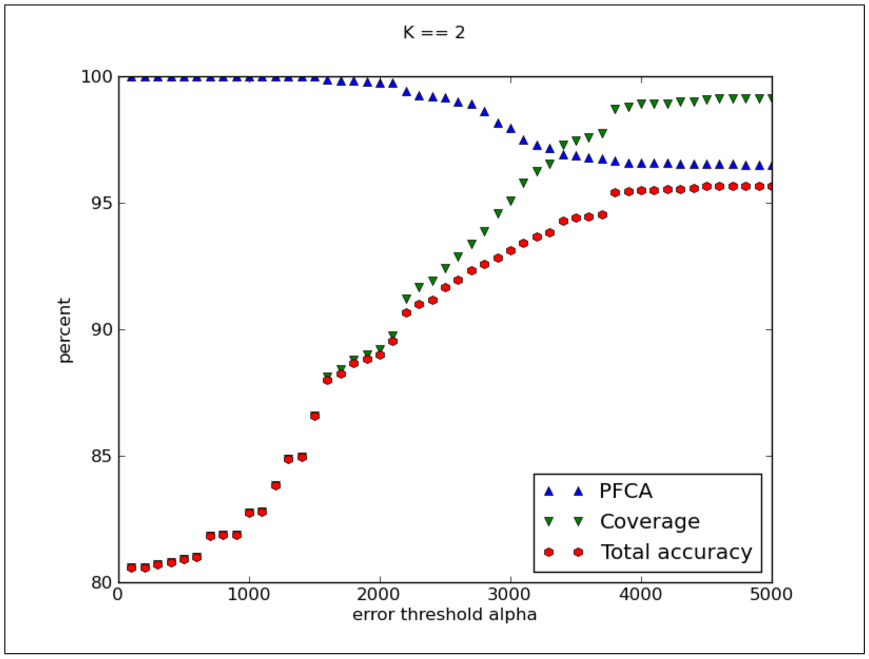
Predicted Font Classification Accuracy (PFCA): the measure of accuracy for the font cluster predictions made for letter's whose distance from any font classifier is within α.

Total Accuracy: defined as the product of assignment prediction and coverage.

4. DISCUSSION

Two types of OCR errors can have visible adverse consequences for a reconstructed font since our technique is applied after OCR. The first type of error the assignment of the wrong character label, because the OCR engine uses a statistical language model that swaps characters class when it makes probabilistic sense based on the letter's neighboring letter's class. This technique is robust to this type of error due to the greedy initial alphabet selection, tokens that represent the greatest amount of letters are given priority over tokens that represent fewer letters.

The second type of error is the recognition of extra, fictitious space. This error is far more prevalent and, unfortunately, much more damaging to the quality of a reconstructed font since the word level co-occurrence is how the original reconstructed font cluster's alphabet is selected. If every time a 'c' of a particular font is



(a) K=2

Figure 2. There is an interesting trade-off between Coverage and PFCA; PFCA does not go below 100 percent until around a α value of 2000, while coverage increases in that interval by roughly 10 percent.

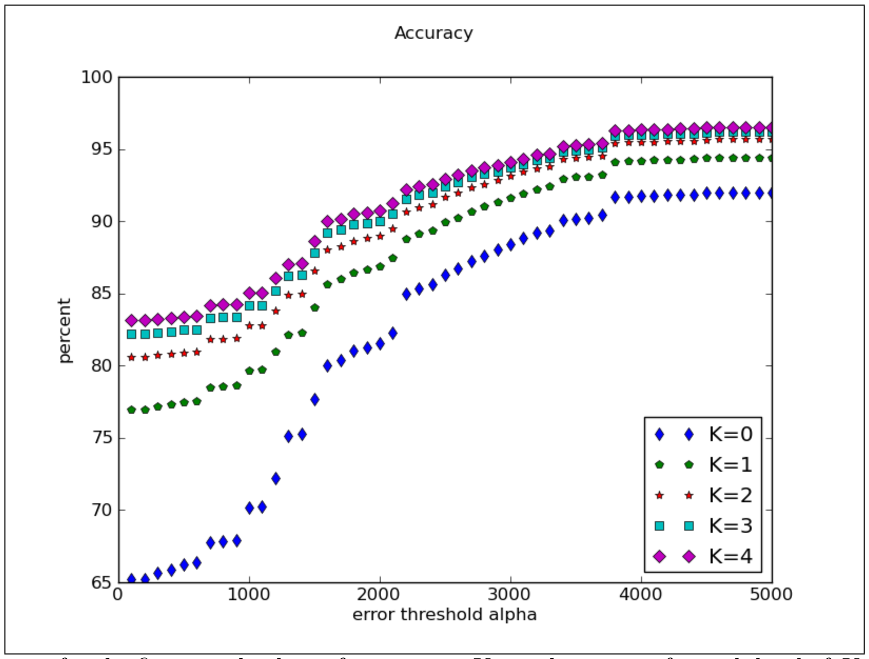


Figure 3. Total accuracy for the five tested values of parameter K, total accuracy for each level of K from Figure 2 are all visible on this graph. The parameterized values that result in maximum total accuracy are α equal to 5000 and K equal to four. For these experiments, increasing k is strongly correlated with increases in total accuracy. This phenomena is explainable because parameter k controls the number of surrounding letters to a token in a reconstructed font's alphabet are also considered part of the same font. The effect on total accuracy is greatest when the parameter K, value increases from zero to one. Setting parameter K to a value greater than zero activates the proximity labelling algorithm. This ensures that letters next to a token known to be in a font cluster is considered part of the same font cluster.

recognized with an extra space on either side of the letter, the reconstructed font representing this "c"'s font will not include the 'c' because the letter will not not co-occur in a word with any other letter. Unfortunately, with the current implementation, the token(s) representing this 'c' will not be classified as belonging to a reconstructed font cluster because none of the reconstructed fonts will contain a similar enough image. A potential fix to this issue would be to modify the automatic font clustering technique to consider single letters next to words as co-occurrences in the token co-occurrence graph.

5. CONCLUSION

Many people across the developing world pay a high premium for a shared Internet connection in order to become interconnected with the rest of the world. Often these connections are extremely low bandwidth, and therefore a requirement for globally accessible digital archives is a small file size for each digital novel. We hope a font reconstruction system can meet the requirements of a world class digital archive, both acessibility and compression.

The Mixed Raster Content (MRC)[10] format, such as DjVu[11–14] and Google Books,[15] serves as inspiration for font reconstruction. Font reconstruction is being developed for the Decapod project, which is a open source system to capture books for digital libraries. The future work for font reconstruction is to combine reconstructed fonts with text image segmentation for a full MRC document. These digital documents should meet the criteria for inclusion in a world class digital archive because they are being developed to be compressed, high quality representations of the original document.

REFERENCES

[1] Michael P. Cutter, Joost van Beusekom, F. S. T. M. B., "Automatic font clustering for font reconstruction," in [*ACM Symposium on Document Engineering (Accepted for publication)*], ACM (2010).

[2] Öztürk, S., Sankur, B., and Abak, A. T., "Font clustering and cluster identification in document images," *Journal of Electronic Imaging* **10**(2), 418–430 (2001).

[3] Carlos Avilés-Cruz, Juan Villegas, R. E.-P., "Unsupervised font clustering using stochastic version of the EM algorithm and global texture analysis," in [*Progress in Pattern Recognition, Image Analysis and Applications*], 3–25 (2004).

[4] Martin Solli, R. L., "Fyfont: Find-your-font in large font database," in [*Image Analysis*], 432–441 (2007).

[5] "FyFont, a visual search engine for fonts." http://media-vibrance.itn.liu.se/fyfont/.

[6] Khoubyari, S. and Hull, J. J., "Font and function word identification in document recognition," *Comput. Vis. Image Underst.* **63**(1), 66–74 (1996).

[7] "The OCRopus(tm) open source document analysis and OCR system." http://code.google.com/p/ocropus/.

[8] Breuel, T., "The OCRopus Open Source OCR System," in [*Proceedings of the Document and Retrival XV, IS&T/SPIE 20th Annual Symposium 2008*], Yanikoglou, B. and Berkner, K., eds., SPIE (2008).

[9] Breuel, T., "Recent progress on the OCRopus OCR system," in [*MOCR'09: Proceedings of the International Workshop on Multilingual OCR*], 1–10, ACM, New York, NY, USA (2009).

[10] Queiroz, R. D., Buckley, R., and Xu, M., "Mixed raster content (MRC) model for compound image compression," in [*Proc EI 99, VCIP, SPIE*], 1106–1117 (1999).

[11] Haffner, P., Bottou, L., Howard, P. G., and Lecun, Y., "Djvu: Analyzing and compressing scanned documents for internet distribution," in [*In Proceedings of the International Conference on Document Analysis and Recognition*], 625–628 (1999).

[12] Bottou, L., Haffner, P., Howard, P. G., Simard, P., Bengio, Y., and Le Cun, Y., "High quality document image compression with DjVu," *Journal of Electronic Imaging* **7**(3), 410–425 (1998).

[13] Mikheev, A., Vincent, L., Hawrylycz, M., and Bottou, L., "Electronic document publishing using DjVu," in [*Proceedings of the IAPR International Workshop on Document Analysis (DAS'02)*], (August 2002).

[14] Le Cun, Y., Bottou, L., Haffner, P., Triggs, J., Riemers, B., and Vincent, L., "Overview of the DjVu document compression technology," in [*Proceedings of the Symposium on Document Image Understanding Technologies (SDIUT'01)*], 119–122 (April 2001).

[15] Langley, A. and Bloomberg, D. S., "Google Books: Making the public domain universally accessible," in [*Proceedings of SPIE Volume 6500, Document Recognition and Retrieval XIV*], 1–10 (2007).