Enhancing Degraded Document Images via Bitmap Clustering and Averaging

John D. Hobby

Bell Labs, Lucent Technologies
Murray Hill, New Jersey 07974

Tin Kam Ho

Bell Labs, Lucent Technologies
Murray Hill, New Jersey 07974

Abstract

Proper display and accurate recognition of document images are often hampered by degradations caused by poor scanning or transmission conditions. We propose a method to enhance such degraded document images for better display quality and recognition accuracy. The essence of the method is in finding and averaging bitmaps of the same symbol that are scattered across a text page. Outline descriptions of the symbols are then obtained that can be rendered at arbitrary resolution. The paper describes details of the algorithm and an experiment to demonstrate its capabilities using fax images.

1 Introduction

In document image processing one often encounters text images of degraded quality that could have resulted from low resolution scanning due to the demands of high-speed, large volume processing or low-bandwidth transmission. Degraded images are not only unpleasant to view on a display device, but they also pose serious challenges to OCR devices whose accuracies are well-known to depend critically on image quality [12].

We attempt to improve the quality of degraded text images by image matching techniques. In essence, we find images of the same symbol occurring over a text page, compute an average outline from the matched bitmaps, and replace all occurrences of that symbol with it. The motivation is that much of the random noise introduced in the scanning and transmission processes can be canceled out with bitmap averaging, so that the resultant images have smooth outlines and much improved appearance.

Dividing the character images into clusters each containing only a single symbol, in a particular font and type size, is a major difficulty. Moreover, few assumptions should be made about the contents of the document. We assume only that the symbol set used in the text is not too large. Finally, to be useful in large-volume processing or fax transmission, the algorithm should be reasonably fast.

Figure 1 shows our top level algorithm. Given an input page as a binary image, we first perform skew detection and page layout analysis up to character segmentation [1] [9]. The bitmaps of the segmented character images are then sent to a clustering procedure, which produces an initial set of clusters. Bitmap averaging, by smooth shading as in [5, 7], is then applied to each of these clusters to obtain an outline character that represents the cluster. Individual character bitmaps are then compared to the cluster average, and mismatches are rejected. The rejected images are compared to other cluster averages to find possible matches. After this, all cluster centers are compared against each other to find possible merges. Finally, the outline characters are rasterized and inserted into the corresponding places in the text image. This final rasterization can be at a higher resolution than the original input.

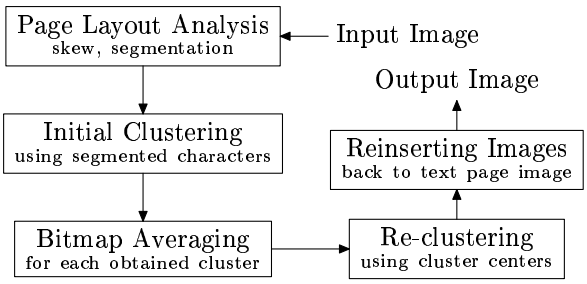


Figure 1: The top-level algorithm

In the following sections we describe the details of the method, with emphasis on clustering and bitmap operations, and an experiment with fax images to evaluate the results. However, the applicability of the algorithm is not limited to degradation caused by fax transmissions.

2 Clustering of Character Images

A procedure is needed to find images of the same character symbol that are scattered on a document page. The images believed to be equivalent are to be averaged and replaced.

In addition to the symbol identity, the images must be matched in point size and typeface. Incorrectly matched images of different fonts are especially undesirable when the resultant page image is to be viewed by human eyes. This

poses a challenge to the matching algorithms: they have to be robust enough to tolerate image noise, but they must also be able to discriminate between minor size and shape variations due to the input point size and typeface. Thus the matching algorithms are more constrained than the shape recognition algorithms used in conventional OCR or the clustering algorithms used solely for improving recognition [8].

With the setup of the current experiment, there are about 1500-2000 character images on a typical page. Most conventional hierarchical clustering algorithms run in O(n^2) time. This translates into about 3 minutes per page on an SGI Challenge XL machine with 150MHz MIPS R4400 processors. This is too slow for most practical purposes, such as serving in a back-end of a fax machine. Furthermore, for a collection of features measured on various scales and compared by Euclidean distance, it is difficult to determine a meaningful threshold on the distance to extract clusters from the resultant hierarchy.

With these concerns, we employed a sequence of different clustering techniques, each applied to a different set of shape features derived from the character images. The motivation is to progressively divide all characters on a page into groups of decreasing sizes, and delay the uses of more expensive techniques until later stages when the groups are sufficiently small. The techniques used are organized into three stages.

2.1 Sorting by Global Measures

In the first stage, a global measure is used to describe a character image, and all images on a page are sorted by the value of this measure. The images are then separated into groups when there is a sufficient gap in the sorted values. The thresholds on the gaps are determined experimentally. They are relative to the scanning resolution.

We applied the procedure in six passes each using a different image measure. The measures are: image size (`sz`), sum of black pixels (`sb`), image width (`wth`), image height (`hgt`), aspect ratio (`asp`), and black pixel density (`sb/sz`). In each pass the images are sorted only within the groups found from the previous pass. When the input group is large, wide gaps between the sorted values may not exist. In such cases, instead of dividing at gaps, the sorted images are divided into evenly sized groups.

2.2 Merging Equivalent Images

In the second stage we use an algorithm that finds the equivalence classes in a given group of images. The equivalence relation ($\approx$) for two images I_1, I_2 is predefined as follows. Let $d_1 = |\mathtt{sz}(I_1) - \mathtt{sz}(I_2)|$, $d_2 = |\mathtt{hgt}(I_1) - \mathtt{hgt}(I_2)|$, $d_3 = |\mathtt{wth}(I_1) - \mathtt{wth}(I_2)|$, $I_1 \approx I_2$ iff $d_1 < t_1$ and $d_2 < t_2$ and $d_3 < t_3$, where t_1, t_2, and t_3 are preset thresholds.

The equivalence classes are found using Eardley's `eclazz()` algorithm [11, Section 8.6]. Eardley's algorithm forms an equivalence relation by computing the transitive closure of the $\approx$ relation. It takes $O(n^2)$ time, but this is affordable at this stage because of smaller group sizes resulting from the previous stage.

2.3 Clustering with Shape Features

The third stage uses a conventional hierarchical clustering algorithm (complete-linkage) with integer-valued feature vectors compared by Euclidean distance. We use five different feature vectors borrowed from the literature in character shape recognition. Each of these vectors is computed by a corresponding feature extraction algorithm that takes a size-normalized character image as input. Since most of the size matching has been performed in the first and second stages, here we can safely ignore the information loss due to size normalization. We normalize each character image to 16×16 pixels while preserving the aspect ratio. The five sets of features used are as follows.

1. *histogram* (`hist`): a concatenation of the vertical and horizontal projection profiles taken on four half images [4]. The vector has $16 \times 4 = 64$ integer components; each in the range [0,8].

2. *contour* (`cntr`): distances from the bounding box to the character's outer contour [4]. The vector has $16 \times 4 = 64$ integer components; each in the range [0,16].

3. *pixel correlation* (`pixcor`): conjunctions and disjunctions of neighboring pixels in various directions [3]. The feature vector has 268 binary components.

4. *subsamples* (`subs`): results of subsampling the normalized image down to one-fourth of the normalized size, and then repeating the process until the image is reduced to one single pixel. In each pass the algorithm replaces each 2x2 window by the sum of the pixel values in it. The vector has 85 integers in [0,256].

5. *stroke direction distribution* (`dirdist`): counts of pixels labeled by the direction of the longest black run they belong to [10]. The vector has 64 integer components in the range [0,16].

2.4 Merging Singletons

There is no backtracking in each of the three preceding clustering stages, so that once two images are separated into different clusters, they remain so in each subsequent steps. At the end of the third stage, a crude attempt is made to merge any single images that do not belong to any existing cluster. This uses the same equivalence finding algorithm as in the second stage, but the equivalence is defined on the size-normalized images, and two images are considered equivalent if their Hamming distance is less than a pre-set threshold.

Table 1 details the outcomes of each clustering step for an example page, and Figure 2 shows images of some clusters extracted from this page.

Table 1: Results of clustering steps for a sample page.

procedure	# gps	avg size	procedure	# gps	avg size
input	1	1555	equivalence	98	15.9
sort (sz)	22	70.7	cluster (hist)	119	13.1
sort (sb)	35	44.4	cluster (cntr)	223	7.0
sort (wth)	42	37.0	cluster (pixcor)	229	6.8
sort (hgt)	44	35.3	cluster (subs)	253	6.1
sort (asp)	50	31.1	cluster (dirdist)	295	5.3
sort (sb/sz)	50	31.1	merge singletons	287	5.4

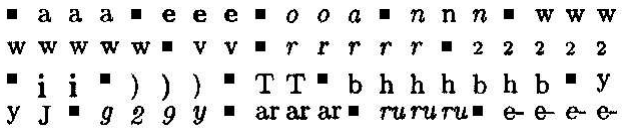


Figure 2: Example clusters extracted from a 200 dpi text page. Members of different clusters are separated by a black square. There are font confusions and mismatched symbols. Also, symbols in a cluster may not be well-segmented characters.

3 Improving the Clustering

Once we have clustered the character images, we can use bitmap averaging to generate a good set of outlines for each cluster. We now improve the clustering via pairwise shape comparisons involving these good outlines. The idea is to ensure that every character image in a cluster matches the outlines for that cluster and that the total number of clusters is as small as possible. This leads to a three stage process:

1. Refine each cluster so that each character image in the cluster matches the outlines for that cluster. This can produce singleton clusters.

2. For each singleton cluster, try to find a non-singleton cluster whose outlines it matches. Then merge the singletons into the selected clusters.

3. Merge pairs of clusters whose outlines match.

Sections 3.1, 3.3 and 3.4 each consider one stage of this process, and Section 3.2 explains how to decide if a character image matches a set of outlines.

3.1 Refining a Cluster so its Character Images Match

Suppose we have a procedure for deciding whether a character image matches a set of outlines, and we want to use it on a cluster of character images such as those in Figure 3a. In this example, 31 n's and 14 u's from an upright font got clustered with 3 n's from a slanted font. Since most of the images were n's, bitmap averaging produces an "n" shape as shown in Figure 3b. Now test all of the character images against the average shape from Figure 3b. Of course the matching procedure will not be perfect—in this example, 26 of the upright n's are declared to match and everything else does not match. Since the non-matching character images are mostly u's, their average is a "u" shape as shown in Figure 3d.

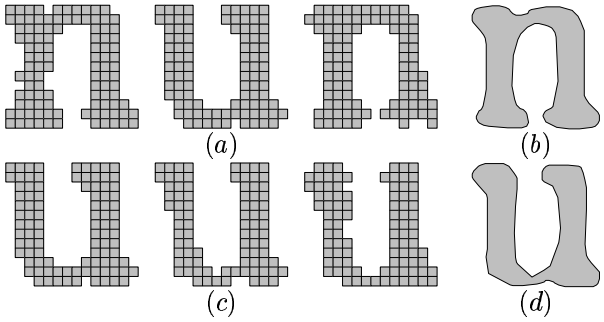


Figure 3: (a) some of the 48 character images in the cluster to be refined; (b) the average of these character images; (c) some of the 22 character images that do not match; (d) the average of the non-matching images. The character images are from a 200×200 dpi fax page.

One could imagine many rounds of reaveraging rejected character images, but this is seldom possible in practice. Matching Figure 3d against the 22 responsible character images produces 11 rejects, but only one of these 11 rejects matches their average bitmap. At this point, we just give up and output the 11 character images as singleton clusters. Hence the cluster of 48 n's and u's got split into a cluster of 26 n's, a cluster of 11 u's, and 11 singleton clusters. Algorithm 1 summarizes this process.

Algorithm 1 How to refine a set $\mathcal{C}$ of character images so each matches the average.

1. Apply bitmap averaging to the bitmaps in set $\mathcal{C}$ to yield outlines A, and let n be the number of bitmaps in $\mathcal{C}$.

2. Find all bitmaps in $\mathcal{C}$ that match A, then output them as a new cluster and remove them from $\mathcal{C}$. If the number of bitmaps remaining in $\mathcal{C}$ is at most $\min(n-2, \frac{9}{10}n)$, go to Step 1.

3. Output each remaining bitmap in $\mathcal{C}$ as a singleton cluster.

3.2 Matching a Bitmap against Outlines

What about the procedure for comparing a character image to the outlines for a shape such as Figure 3b? We could require the bitmap to match the result of rasterizing the outlines, but this is too restrictive. Therefore, we expand the outlines, then rasterize and compare with original bitmap. It is also possible to rasterize contracted versions of the outlines or to expand or contract the black areas in the bitmap. For best results, we have to be prepared to combine all of these ideas.

First consider rasterizing an expanded version of the outlines. If S is the region described by the outlines, we can try rasterizing

$$S \oplus B_\epsilon = \{\, s + b \mid s \in S \text{ and } b \in B_\epsilon \,\}, \qquad (1)$$

where ϵ is a constant to be chosen later, B_ϵ is the set of points with $\max(|x|, |y|) \leq \epsilon$, and $\oplus$ denotes the Minkowski sum. Letting S be the region in Figure 3d causes (1) to look like Figure 4a. Rasterizing this by darkening the pixels whose centers are in (1) produces Figure 4b: the less restrictive matching condition is that the black pixels must be a subset of those in Figure 4b.

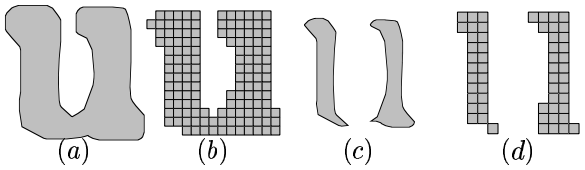


Figure 4: (a) The region $S \oplus B_{1/2}$; (b) the rasterization of this region; (c) the region $S \ominus B_{1/2}$; (d) the rasterization of this region.

In order to check for white pixels in the wrong place, we need to rasterize contracted versions of the outlines. The contraction operation for a region S is

$$S \ominus B_\epsilon = c(c(S) \oplus B_\epsilon),$$

where $c(\cdot)$ denotes the set complement. This shrinks black areas, reducing Figure 3d to Figure 4c. Turning on pixels whose centers lie inside produces the rasterization shown in Figure 4d. We can now say that a bitmap matches Figure 3d if pixels are black whenever they are black in Figure 4d and white whenever they are white in Figure 4b.

To implement this test, we use the techniques of Guibas, Ramshaw, and Stolfi [2] to compute outlines for $S \oplus B_\epsilon$ and $S \ominus B_\epsilon$. In practice, it helps to add conditions involving expanded and contracted version of the bitmap. This copes with disappearing strokes such as the bottom of the "u" in Figures 4c–d.

3.3 Matching Singletons with other Clusters

Since Algorithm 1 can produce a lot of singletons, it is important to try to match them with existing clusters before giving up and leaving them unmatched. This is a fairly simple matter of using the techniques of Section 3.2 to compare the character bitmap for each singleton cluster to the averaged outlines for each non-singleton cluster. There are a lot of possibilities to consider, but most of them can be thrown out quickly by just comparing the bounding box of the character bitmap to that of the outlines it is supposed to match.

If a character bitmap C does not match the outlines for any possible cluster, we try again with slightly higher tolerances. The idea is that there probably is some cluster that C is supposed to match, and we can locate a good candidate by finding the "best" match for the higher tolerances.

What do we mean by "best" match? The matching procedure from Section 3.2 works by requiring pixels from one bitmap to obey constraints based on another bitmap. When there is a violation, counting the violating pixels provides a quantitative measure of the degree of mismatch. Hence if more than one cluster provides an adequate match for C, we can choose the one that produced the fewest violating pixels when compared using the smaller tolerances.

3.4 Merging Clusters whose Averaged Outlines Match

Handling singleton clusters explained in Section 3.3 substantially reduces the number of clusters (from 710 to 385 for a typical 200dpi test page), but there is still a lot of room for improvement. The reduced cluster count makes it more practical to apply a quadratic algorithm based on a comparison function that examines the averaged outlines for two clusters and decides if they match. We again use Eardley's `eclazz()` algorithm [11].

Suppose A and A' are the outlines to be compared and let $\bar{A}$ and $\bar{A}'$ be the corresponding point sets. (Just the outlines, not their interiors). We can say that A and A' match to within ϵ_0 if each vertex of A is within euclidean distance ϵ_0 of a point in $\bar{A}'$ and each vertex of A' is within distance ϵ_0 a point in $\bar{A}$.

If there are m vertices in A and n vertices in A', the "match to within ϵ_0" test could involve mn evaluations of the distance between a point and a line segment, but careful pruning based on bounding boxes gives a substantial speedup.

If A matches A' and A' matches A'', Eardley's algorithm assumes that A matches A'' or at least it is safe to cluster A and A'' together. The "match to within ϵ_0" test is not fully compatible with this idea, but it seems to work well in practice as long as ϵ_0 is less than one pixel unit.

Alternatively, the outlines for each cluster can be rasterized and re-clustered using the shape feature vectors as before, but with much tighter distance thresholds when groups are extracted from the cluster hierarchy. The most useful shape features in this step include the contours, the histograms, and stroke direction distribution.

4 Results

We implemented the complete algorithm in a mixture of C and C++ and tested it on 28 fine mode fax pages (200×200 dpi) and 20 standard mode fax pages (200×100 dpi). The test pages were photocopied once before faxing, and then the receiving fax machine captured the image electronically.

The running times averaged about 50 seconds per page on a 200Mhz SGI Challenge L using one of the eight Mips R4400 processors. Approximately 22% of the time was spend in the initial clustering, 61% was spent improving the clusters (Sections 3.1–3.3), and 17% was spent in final merging process (Section 3.4).

4.1 Evaluating the Clustering Process

It is important to classify the character images into as few clusters as possible, since this is how the algorithm achieves its benefits. Yet it is even more important to avoid clustering incompatible character images since this leads to "mistakes" in the output; e.g., averaging 34 n's with 14 u's produces an

"n" shape that would erroneously be used to represent the 14 u's. We refer to such erroneous character images as *misspellings* or *font-substitution errors* depending on the type of incompatibility. Using an "n" shape to represent a "u" is a misspelling. An example of a font-substitution error would be representing an italic "n" by an image constructed primarily from roman n's.

In order to evaluate misspellings and font-substitution errors, the test pages were matched with ground truth as explained in [6]. Thus each object that page layout analysis identified as a character was labeled with one or more (font name, character identity) pairs or was labeled as junk or as containing a partial character. Some of the images labeled as junk actually involved text material such as header lines generated by the fax machine. For this reason, no misspellings were charged for including junk images in the clusters of character images.

Table 2 describes the test pages and gives statistics about the characters to be averaged. The reason for the large numbers of partial characters in the 200×100 dpi data (second half of the table) is that page layout analysis could not cope with the tendency of thin horizontal strokes to drop out.

Document			Average number of characters				
Id	Font	Pages	True	J	P	M	N
A	cmr 10	5	1715	76	11	79	1515
B	Times 10	6	2371	71	26	140	2032
C	cmr 12	5	1272	67	12	11	1235
E	cmr 10	4	1879	70	31	73	1699
G	cmr 11	4	1643	83	70	69	1427
S	cmr 11	4	1453	194	39	58	1165
A'	cmr 10	4	1698	73	231	53	1448
B'	Times 10	4	2243	67	95	75	2021
C'	cmr 12	3	1511	65	403	8	1303
E'	cmr 10	1	2138	63	400	49	1827
G'	cmr 11	4	1643	112	396	53	1327
S'	cmr 11	4	1453	76	288	49	1203

Table 2: Statistics about the test pages and the character images extracted by page layout analysis. Documents A–S are 200×200 dpi faxes, and documents A'–S' are 200×100 dpi faxes. The "True" column lists the average characters per page from the ground truth and columns J, P, M, N, list the junk, partial, merged, and normal characters among what page layout analysis identified as characters.

Table 3 shows the number of clusters obtainable and how many errors occur. Some misspellings were deemed "minor" because the characters involved are essentially identical; e.g., period versus centered dot "·", comma versus baseline single quote, "l" and the digit "1" in the Times Roman.

Table 3 shows that the misspellings can be controlled by setting the tolerance small enough, but it is difficult to eliminate the font-substitution errors. Further work is needed to see how the matching functions in Sections 3.2 and 3.4 can be made more sensitive to this. The 200×100 dpi test pages (lower sections of the table) show that tighter tolerances are needed in this case, and the actual number of clusters is further from the ideal value.

The ideal cluster counts in the table are estimates based on the ground truth for those character images not listed as "junk", "partial", or "merged." Other characters were grouped as in the actual clusters. The columns labeled "differentials" are estimates based on the number of clusters that result from using the ground truth to refine the actual clusters so as to eliminate all the misspellings and font-substitution errors. The "More" column is the amount by which this exceeds the "Ideal" cluster count. It tells how much the cluster count could have been reduced by better clustering. This is a fairly small fraction of the ideal cluster count for the 200×200 dpi pages, but it gets a little larger if we go to 200×100 dpi or tighten the tolerances to get rid of all the spelling errors. By listing most of the problems with more than one ϵ value, the table gives an idea of the trade off between the goals of reducing the cluster count and avoiding errors.

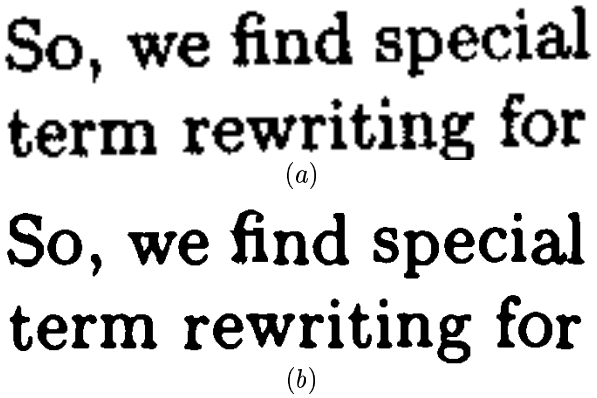
		Errors			No. clusters		Differentials	
Id	ϵ_1	S_1	S_2	F	Ideal	Act.	More	Less
A	0.55	1.8	1.0	91	219	261	73	31
B	0.55	8.5	1.2	6	239	294	73	18
C	0.55	0.2	0.0	1	127	172	49	4
E	0.55	0.5	0.2	23	227	279	72	20
G	0.55	0.5	0.0	57	292	353	101	40
S	0.55	0.0	0.2	17	268	268	50	50
A	0.3	1.8	0.0	74	243	378	153	18
B	0.3	9.0	0.0	8	273	408	148	13
C	0.3	0.2	0.0	0	130	268	141	3
E	0.3	0.0	0.8	15	253	409	167	11
G	0.3	0.5	0.0	38	304	495	216	25
S	0.3	0.0	0.0	12	298	374	117	41
A'	0.45	1.0	1.5	54	302	524	253	31
B'	0.45	0.5	20.0	10	280	526	273	27
C'	0.45	0.0	1.7	0	256	483	238	11
E'	0.45	0.0	0.0	28	299	645	367	21
G'	0.45	1.2	2.8	35	434	722	322	34
S'	0.45	0.0	1.5	15	360	567	236	29
B'	0.4	0.8	3.2	5	291	697	430	24

Table 3: Misspellings, font-substitution errors and number of clusters per page for alternative values of ϵ. The "S_2" column lists harmful misspellings per page; the "S_1" column lists other misspellings; the "F" column lists font-substitution errors; the "Ideal" column gives an estimate of the ideal number of clusters; the "Actual" column counts the clusters produced by our algorithm; and the "more" and "fewer" columns estimate the number of extra clusters due to inability to match everything that should be matched and the number by which errors mistakenly reduced the cluster count.

4.2 Image Quality

Figure 5 illustrates the image quality before and after using the algorithm to scale a 200×200 dpi fax image to 600 dpi, and Figure 6 gives a similar comparison for a 200×100 dpi fax. In many ways, the images are dramatically improved. Most of the characters in Figure 5b look like they came from a slightly blurry 600 dpi image, and this is true to a lesser extent in Figure 6b. Certain characters such as the "fi" ligature and the capital "S" in Figure 5 do not show as much improvement because it was hard to find other character bitmaps with which to average them. This problem

is somewhat more serious in Figure 6b where many of the characters broke up, leaving partial characters that are not well suited to bitmap averaging. (The uneven baselines in the output image are an artifact caused by imprecise skew correction during page layout analysis).



(a)

(b)

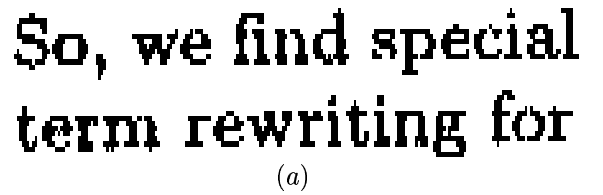
Figure 5: A portion of a 200×200 dpi fax image before and after using the algorithm to scale it to 600dpi.

4.3 Improved Optical Character Recognition

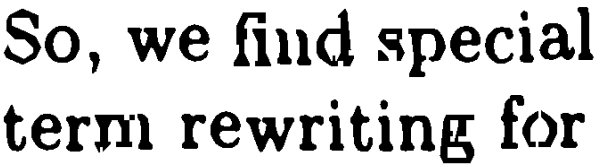
We tested OCR accuracy using Adobe Acrobat Capture Version 1.0. Our algorithm offers two opportunities for improved OCR accuracy: better image quality as shown in Figures 5 and 6, and grouping of character images into clusters that are believed to share a common symbol identity. Taking full advantage of these opportunities is challenging because commercial OCR engines have not been designed to use such information. High quality images from character bitmap averaging are interspersed with inferior images for characters that could not be averaged. If the page image is scaled up to allow accurate representation of the results of bitmap averaging, there is no way to tell the OCR engine that some glyphs are just magnified low-resolution images, even though we know which glyphs these are. In addition, some of the images from bitmap averaging can contain segmentation errors. For instance, one test page contained 9 occurrences of "ar," none of which could be segmented into two characters. Averaging these 9 images together produces a good-looking "ar," except that the characters run together. If rendered at high resolution, the result is an unusual segmentation problem that OCR engines find particularly difficult.

For these reasons, the OCR results summarized in Table 4 used output from the algorithm rendered at 200×200 dpi. Instead of using the algorithm to scale up to 600 dpi as shown in Figures 5 and 6, the 200×200 fax pages were not scaled up and the 200×100 pages were scaled only as necessary to correct the non-square aspect ratio that Acrobat Capture was not prepared to handle.

The percentages in Table 4 are based on the ratio of string edit distance to number of ground truth characters. The string edit distance might be slightly overstated because of the way we used the Unix *diff* command to compute it. The error rates are high because of the low resolution, especially for the 200×100 dpi pages used in the bottom half of Table 4. Another reason for the high error rates is the inability to handle mathematical symbols on the test pages. (There were no displayed formulas).



(a)



(b)

Figure 6: A portion of a 200×100 dpi fax image before and after using the algorithm to scale it to 600dpi.

Id	OCR error rate			% improvement
	input	output	after voting	
A	7.2	6.6	5.9	18
B	4.4	4.8	3.9	11
C	1.8	2.0	1.7	6
E	3.7	2.8	2.3	38
G	13.5	12.0	11.0	19
S	14.1	13.8	11.7	17
A'	17.3	10.7	13.9	38
B'	12.2	8.3	7.7	37
C'	11.0	6.9	6.9	37
E'	20.7	11.1	11.0	47
G'	26.2	19.4	19.3	26
S'	31.5	21.8	20.7	34

Table 4: OCR error percentage rates for each of the test documents for input page images, for output pages rendered at 200dpi, and for the output pages after trying to use the majority vote technique to improve the results. The last column gives the reduction in error rate with the best strategy. All results are for Adobe Acrobat Capture 1.0.

The "after voting" column shows the result of trying to take advantage of the grouping of character images into clusters that are believed to share a common symbol identity. Even though all members of such a cluster are generated from the same outline, differences in context and differences in phase relative to the pixel grid can easily lead to differing OCR results. Correcting for this should be a fairly simple computation involving taking a majority vote among the OCR results for each cluster of character images and correcting the OCR results that do not agree with the majority.

It was quite difficult to do this in practice because OCR engines such as Acrobat Capture do not provide an easy way of associating the OCR output with the corresponding images. Acrobat Capture can produce PDF files that give x, y

coordinates for each character, but the x coordinates do not always match those of the corresponding character images. We created a program that attempts to match the PDF files to the character images, but the effort was only partly successful. For this reason, the "after voting" numbers for the 200 dpi pages (top half of Table 4) probably do not live up to their potential and the "after voting" numbers for the 200×100 dpi test pages do not reliably show any improvement.

5 Conclusions

We have described a method to enhance the quality of degraded text images and applied it to faxes. The resultant images are generally improved in display appearance, and substantial reduction in OCR errors is achieved using a commonly available OCR software.

The clustering results are important side products of the procedure and they have other potential uses that remain to be explored. For instance, if subpixel shifting in character positions can be ignored, all members of a cluster could have the same rasterization and the recognition procedure need only process it once. Using a single representative for each cluster could greatly improve image compression [13]. If a language model can be assumed, the identity of a symbol can be determined using context analysis without very accurate image-based recognition output.

In our experiments we performed all processing one page at a time. Better results could probably be achieved if the algorithm were modified to take in multiple pages of the same document from the same source. After a sufficient number of pages are processed, the clusters could be stable enough that character bitmaps from subsequent pages will need only to be matched to established cluster centers. This will be particularly attractive in large-volume scanning of a single source, such as books, or multiple issues of the same journal, in a digital library application.

Potential improvements in run time remain to be investigated. Many of the steps could be parallelized, and our tests on 8 and 12 processor SGI machines would have benefited. The smooth-shading technique for bitmap averaging is much more compute intensive than the alternatives discussed in [7], and this trade-off should be investigated.

6 Acknowledgements

We thank Henry Baird for interesting discussions and help in interfacing with the page reader. Ken Church also provided significant input. Dar-Shyang Lee's help in capturing the fax images, and Sean Quinlan's help in interfacing with the OCR software, are also much appreciated.

References

[1] H. S. Baird. Anatomy of a versatile page reader. *Proceedings of the IEEE*, 80(7):1059–1065, July 1992. Special Issue on OCR.

[2] L. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. of the 24th Annual Symp. on Foundations of Computer Science*, pages 100–111, 1983.

[3] T. K. Ho. Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, pages 278–282, Montreal, Canada, Aug. 1995.

[4] T. K. Ho and H. S. Baird. Perfect metrics. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, pages 593–597, Tsukuba Science City, Japan, Oct. 1993.

[5] J. D. Hobby. Polygonal approximations that minimize the number of inflections. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 93–102, Jan. 1993.

[6] J. D. Hobby. Matching document images with ground truth. In *ICDAR'97: Fourth International Conference on Document Analysis and Recogntion*, 1997.

[7] J. D. Hobby and H. S. Baird. Degraded character image restoration. In *Proceedings of the Fifth Annual Symposium on Document Analysis and Image Retrieval*, pages 233–245, 1996.

[8] T. Hong and J. J. Hull. Improving ocr performance with word image equivalence. In *Proceedings of the Fourth Annual Symposium on Document Analysis and Image Retrieval*, pages 177–189, 1995.

[9] D. J. Ittner and H. S. Baird. Language-free layout analysis. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, pages 336–340, Tsukuba Science City, Japan, Oct. 1993.

[10] S. Mori, K. Yamamoto, and M. Yasuda. Research on machine recognition of handprinted characters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(4):386–405, July 1984.

[11] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[12] S. V. Rice, J. Kanai, and T. A. Nartker. An evaluation of OCR accuracy. In *Information Science Research Institute, 1993 Annual Research Report*, pages 9–20. University of Nevada, Las Vegas, 1993.

[13] Q. Zhang and J. M. Danskin. Bitmap reconstruction for document image compression. In *International Symposium on Voice, Video, and Data Communications*, 1996.