Matching Document Images with Ground Truth

John D. Hobby

Bell Laboratories
Murray Hill, NJ 07974

Since optical character recognition systems often require very large amounts of training data for optimum performance, it is important to automate the process of finding ground truth character identities for document images. This is done by finding a transformation that matches a scanned image to the machine-readable document description that was used to print the original. Rather than depend on finding feature points, a more robust procedure is to follow up by using an optimization algorithm to refine the transformation. The function to optimize can be based on the character bounding boxes—it is not necessary to have access to the actual character shapes used when printing the original.

1 Introduction

Training and testing optical character recognition systems often requires large numbers of realistic character and word images. If the correct "ground truth" character identities have to be entered or corrected by hand, it is difficult to gather enough accurate data. Semiautomatic methods are popular now, but they still require a lot of labor [8, 9, 10]. Much of this labor can be avoided by automatic ground-truthing where a scanned page image is matched with the original document description that was used to print the page. The procedure suggested by Kanungo [4] is to look for certain feature points in the scanned image, transform the original page description to make the feature points match, and then use template matching to look for character images near where the original page description says they should be.

The goal of this work is primarily to find a more robust way of transforming the original page description to match the image, and secondarily to see if we can avoid the need to have fonts available for template matching.

The original page description identifies each character on the page and gives its bounding box coordinates. Figure 1 shows how Kanungo uses this information to find feature points. The feature points are the bounding box corners where $x+y$, $x-y$, $-x+y$ and $-x-y$ are maximized. Performing a similar computation for connected components in the scanned page image yields another set of four feature points. Kanungo then finds a geometric transformation that aligns the two sets of feature points and hopefully aligns the character bounding boxes with the corresponding character images.

This can work, but it was not very reliable in our tests. Scanned images often have speckles and other material not in the original page description, and this can cause the wrong feature points to be found. Even when this does not happen, transformations based on just a few feature points can give suboptimal matches. For these reasons, we attempt to find a transformation that makes the entire page description fit the image as well as possible. We define a function that measures this fit, and then use standard optimization techniques. Since it is still important to have a good starting point for the optimization, the new approach does not replace feature point analysis, it complements it.

Section 2 defines the optimization problem and explains what standard techniques are best for solving it. Section 3 explains how careful bucketing strategies can speed up the optimization enough to make it practical. Section 4 investigates the problem of assigning ground truth to character and word images without a priori knowledge of the character shapes or non-geometric distortions in the printing and scanning process. Finally, Section 5 gives results and discusses applications, and Section 6 gives some concluding results.

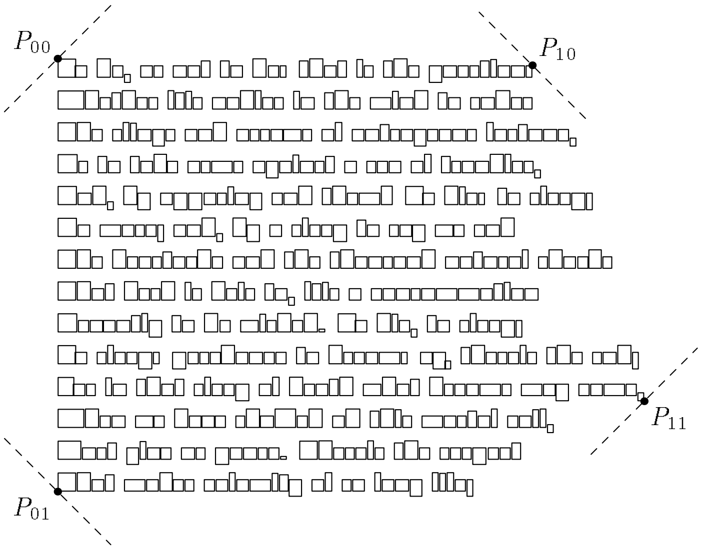


Figure 1: The bounding boxes for all characters on a small sample page with Kanungo's feature points indicated. The feature points are chosen so that the 45° dashed lines are supporting lines.

2 Finding the Transformation

The task is to take character bounding boxes such as those in Figure 1, and find a geometric transformation that matches them to a similar set of bounding boxes derived from the scanned image. Specifically, we use the bounding boxes for 8-connected sets of black pixels in the image and choose an affine transformation

$$T(x, y) = \begin{pmatrix} t_{xx} & t_{xy} \\ t_{yx} & t_{yy} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}. \tag{1}$$

An initial estimate for T can be computed from the feature points illustrated in Figure 1. We need to improve this estimate by minimizing a function that depends on the six parameters that appear in (1), and measures the degree by which the character bounding boxes fail to match the image when the transformation T is applied.

In order to have axis-aligned rectangles, it is best to start by applying T to each connected component in the image and then compute bounding boxes as indicated by dashed lines in Figure 2a. Call these the *image boxes* and refer to the character bounding boxes from the original page description as *ideal boxes*. Since characters can break up and/or merge together during printing and scanning, there is not necessarily a one-to-one correspondence. In the example of Figure 2b, there are two image boxes for the ideal box "r," and ideal boxes "a" and "l" both correspond to the same image box.

2.1 The Mismatch Function

The above example suggests that we measure the degree of mismatch by developing a function d that takes two boxes and measures how much they would have to be changed in order for one of the boxes to contain the other. For each ideal box A, we can find the image box B that minimizes $d(A, B)$, and then apply a standard vector norm to the resulting list of d values. This approach allows characters to break up or merge together and does not penalize for junk or non-text material for which there are image boxes but no ideal boxes.

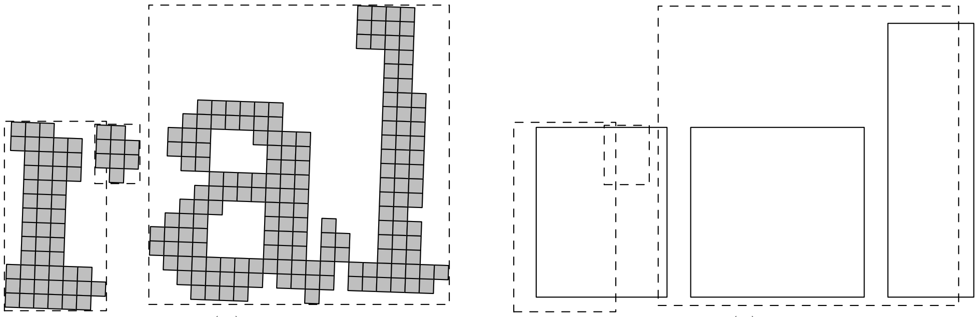


Figure 2: (a) Portions of a slightly rotated 200dpi page image with bounding boxes; (b) the same image boxes (dashed lines) superimposed on the ideal boxes from the original page description (solid lines).

Hence we define

$$\begin{aligned} d(A,B) &= \min\big(d_f(A_{x1},A_{x2},B_{x1},B_{x2}) + d_f(A_{y1},A_{y2},B_{y1},B_{y2}), \\ &\qquad d_f(B_{x1},B_{x2},A_{x1},A_{x2}) + d_f(B_{y1},B_{y2},A_{y1},A_{y2})\big) \\ &\qquad + d_p(A_{x2}-A_{x1},B_{x2}-B_{x1}) + d_p(A_{y2}-A_{y1},B_{y2}-B_{y1}), \end{aligned}$$

where $x1$ and $x2$ subscripts refer to a box's minimum and maximum x coordinate, $y1$ and $y2$ subscripts refer to the minimum and maximum y coordinate, d_p is a penalty term that is nonzero when the ratio of its arguments is too small or too large, and

$$d_f(x_1,x_2,x_3,x_4) = \begin{cases} 0 & \text{if } x_3 \le x_1 \text{ and } x_2 \le x_4; \\ \min(|x_3-x_1|,|x_4-x_2|) & \\ \quad + \max(0, x_2-x_1-(x_4-x_3)) & \text{otherwise.} \end{cases}$$

Function d_f is the amount of translation and stretching necessary to fit interval $[x_1,x_2]$ within interval $[x_3,x_4]$. The d_p terms ensure that unreasonably large objects are not treated as run-together characters and that broken characters are required to contain reasonably large connected components. A generous choice is

$$d_p(a,b) = \max(0,\ \max(a,b) - 8\cdot\min(a,b)).$$

We can now define the mismatch function as follows: take the six parameters that appear in (1); use them to find bounding boxes of the connected components in the transformed image, chooses among these transformed image boxes, an image box B that minimizes $d(A,B)$ for each ideal box A; and apply a vector norm to the resulting $d(A,B)$ values. We choose the L_4 norm (fourth root of sum of fourth powers) since experiments suggested that the L_∞ norm is too sensitive to noise and the L_2 norm causes the mismatch function to have too many local minima.

2.2 Minimizing the Mismatch

Most standard optimization methods do not work well on the mismatch function because it lacks continuity and smoothness properties. Many comparisons are needed to evaluate the function and it does not take much of a change in the transformation parameters (1) to reverse one of the comparisons. This means that the mismatch function has an extremely large number of slope discontinuities. As explained by Wright [13], optimization problems of this kind are best solved by direct search methods such as Nelder-Mead [7] and Torczon's algorithm [12].

Torczon's algorithm is well suited to parallel computation and it has guaranteed convergence properties that Nelder-Mead lacks, but the results in Section 5 will show that it requires significantly more function evaluations. In addition, it is hard to get enough information about the mismatch function to make use of the convergence properties.

For these reasons, we concentrate on Nelder-Mead. For our six-dimensional problem, the Nelder-Mead algorithm maintains a set of seven

$$\left(t_{xx}, t_{xy}, t_{yx}, t_{yy}, \frac{t_x}{W}, \frac{t_y}{H}\right)$$

values that are the vertices of a simplex in Euclidean 6-space, where W and H are constant scale factors. (It suffices to let W and H be the width and height of the page). If T_7 is the vertex where the mismatch function is greatest and $\bar{T}$ is the average of the other six vertices, a Nelder-Mead step consists of trying one or two points along the $\bar{T}\,T_7$ line and trying to update the simplex by replacing one of the existing vertices. In the uncommon case where this gives no improvement, the simplex is contracted by replacing each simplex vertex T_i with $(T_i + T_1)/2$, where T_1 is the simplex vertex for which the mismatch function is lowest.

The usual convergence test is to see if the simplex vertices or the function values at the vertices are all close together. We modify this by defining a *critical value* for the mismatch function, and using very tight tolerance in the other tests if the critical value has not been achieved. The purpose of the critical mismatch value is to ensure that there is a good chance of assigning appropriate ground truth. The critical value is what the mismatch function would return if each $d(A, B)$ value were half the median character width.

3 Efficient Implementation

The Nelder-Mead algorithm can require 300 function evaluations to find the transformation parameters that minimize the mismatch function. For a typical page containing 1800 characters and 1800 connected components, Section 2.1 apparently requires 1800^2 evaluations of $d(A, B)$ for each invocation of the mismatch function. The resulting 1 billion $d(A, B)$ evaluations would probably make the overall algorithm unacceptably slow.

Two strategies result in significant speed-ups:

1. Use a carefully chosen subset of the ideal boxes—the ones where $d(A, B)$ is likely to be largest if the transformation parameters are wrong.
2. Preprocess the transformed image boxes using a bucketing algorithm that allows $d(A, B)$ to be evaluated only for the most reasonable (A, B) pairs.

Strategy 1 is simplest, but it needs to be used carefully since it involves changing the mismatch function rather than just evaluating it more efficiently.

3.1 Choosing a Subset of the Ideal Boxes

The mismatch function is designed to allow for material in the page image that does not correspond to any characters in the original page description. Hence it is natural to try to simplify the problem by basing the mismatch function's ideal boxes on just a subset of the characters. The shaded boxes in Figure 3 are an example of such a subset.

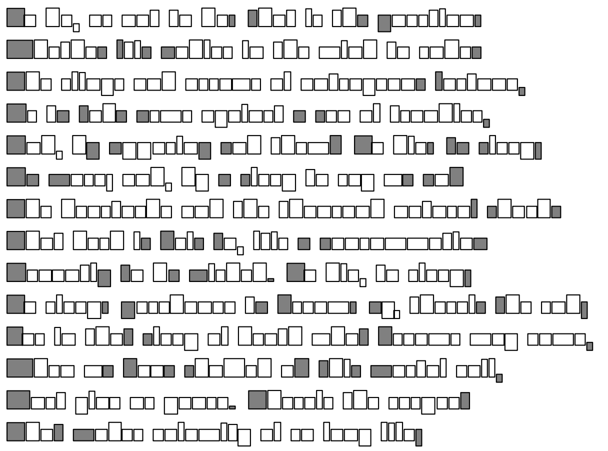


Figure 3: The character boxes from a sample page with a subset shaded to indicate which of them might be used as ideal boxes for the mismatch function.

The selected characters in Figure 3 are at the beginnings and ends of lines, around any other large empty spaces, and around some of the word breaks. If a poorly chosen transform is applied to the image, it is likely that many of the selected boxes will not match any of the resulting image boxes. Hence the mismatch function will tend to return large values when given non-optimal transformation parameters.

It is easy to choose such a subset of the characters if we assume that the original page description lists boxes approximately in reading order. Suppose there are n characters numbered 0, 1, 2, ..., $n-1$, where each character i has bounding box A_i. Let σ be a pseudorandom function that maps 0, 1, ..., $n-1$ into the interval $[1, 1.3]$, and consider the positive integers less than n in order of decreasing

$$\sigma(i) \cdot d(A_{i-1}, A_i).$$

For each such integer i, make i and $i-1$ part of the chosen subset until the chosen subset reaches some predetermined size. This predetermined size will typically be something like

$$\min(\max(300, 0.15n), n).$$

The purpose of the function σ is to ensure that if there are a lot of roughly equal $d(A_{i-1}, A_i)$ values, the chosen subset samples them in a roughly uniform manner with respect to the index i.

3.2 Bucketing Strategies

Transformed image boxes B_0, B_1, ..., B_{m-1}, must be preprocessed so that for any given box A, we can quickly find the B_i that minimizes $d(A, B_i)$. It is natural to use bucketing because the B_i are distributed in a fairly uniform fashion across the non-blank areas of the page, and we can find a reasonable upper bound on the optimal $d(A, B_i)$. This is because the ideal boxes are easily ordered so that if A' follows A, the optimal B_i for A is likely to produce a low value for $d(A', B_i)$ as well.

Another way to guess a good B_i is to use information from a previous invocation of the mismatch function. If the transformation T has not changed much, the transformed image boxes B_0, B_1, ..., B_{m-1} will not differ much from one invocation to the next. Hence we can try the B_i that corresponds to the one chosen last time for ideal box A. Thus we have two guesses for image boxes that are supposed to give low $d(A, B_i)$ values and these provide an upper bound on the actual minimum d value.

There are many ways to set up buckets so that a good upper bound on the minimum d value and a sufficiently uniform distribution of image boxes sharply limit the number of buckets and image boxes to be examined. One method is to classify boxes according to (x_1, y_1) and

$$\max\left(\frac{x_2 - x_1}{W}, \frac{y_2 - y_1}{H}\right), \tag{2}$$

where W and H are the width and height of the page and (x_1, y_1) and (x_2, y_2) are the minimum and maximum x, y coordinates for the box. The observed values of (2) are grouped so that one group has the small values and each other group covers at most a factor of two. Within each group, the bucket is determined by (x_1, y_1) in the natural fashion. Each time we go from one group to the next, values of (2) double and we reduce the number of (x_1, y_1) buckets by a factor of four.

4 Assigning Ground Truth

We have seen how to find a transformation (1) that makes the image boxes approximately match the ideal boxes that carry ground truth character identities from the original document description. If this were an exact one-to-one correspondence, it would be trivial to assign ground truth to the image boxes. When the correspondence is far from one-to-one, the most reliable approach is probably Kanungo's procedure of comparing against the expected character shapes with a small range of x and y displacements. The purpose of this section is to consider what can be done if the ideal character shapes are not available.

This pleasant situation of a one-to-one correspondence is most relevant to the problem of assigning ground truth on a word-by-word basis, so we start with this problem. We then use information from the word-by-word problem to attack the more difficult problem of assigning character-level ground truth.

4.1 Ground Truth at the Word Level

Consider the problem of identifying words in the original document description with the corresponding parts of the page image. We can assume that a word is described by consecutive records from the document description, where each record gives a character identity and the corresponding bounding box. Hence the task is to find word breaks in the document description, combine character bounding boxes to get a bounding box for each word, and then expect the transformed image boxes for each word to be approximately contained in the word's bounding box.

One way to decide if there should be a word break between two records in the document description is just to measure the distance between the character bounding boxes and test it against a threshold as Chen et. al. do [1]. In our case, the original document description was derived from TeX output so we use a version of Knuth's rule for finding word breaks in TeX output [5, 6]. Insert a word break between document description records A and B if

$$B_{x1} - A_{x2} > \frac{s}{6} \quad \text{or} \quad B_{x1} - \bar{x} < -\frac{2s}{3} \quad \text{or} \quad |B_{y2} - A_{y2}| > \frac{5s}{6},$$

where $x1$ and $x2$ subscripts refer to the minimum and maximum x coordinates, $y2$ subscripts refer to the bottom y coordinate, s the average of A's font size and B's font size, and $\bar{x}$ is the maximum of A_{x2} and the C_{x2} values for all records C that precede A and follow the last word break. Another way to say this is that we break the document description into words by checking for horizontal spaces of more than $\frac{1}{6}$ of the font size or vertical spaces of more than $\frac{5}{6}$ of this size, while allowing $\frac{2}{3}$ of this size for backspacing when building up accented characters.

Refer to the bounding box of all the ideal boxes in a given word as the *word box*. We can assign transformed image boxes to words by just testing for approximate containment. An image box B is approximately contained in word box W if

$$d_f(B_{x1}, B_{x2}, W_{x1}, W_{x2}) + d_f(B_{y1}, B_{y2}, W_{y1}, W_{y2}) \leq \epsilon, \tag{3}$$

where d_f is as given in Section 2.1 and ϵ is a suitable threshold. This test is fast enough that we can try each transformed image box against each word box until finding a word box that approximately contains the image box.

4.2 Character Level Ground Truth

If we want to find the portion of the page image that corresponds to each character in the original document description, the word-level ground truth allows the problem to be solved separately for each word. The correspondence between ideal boxes and transformed image boxes may be imprecise as shown in Figure 4a, but the situation can by improved by doing an additional transformation specific to the current word. The bounding boxes of the transformed image boxes assigned to the word (dashed lines in Figure 4b) will not precisely match the word box (solid lines in Figure 4b), but a transformation of the form

$$\bar{T}(x, y) = (\bar{t}_{xx}x + \bar{t}_x,\ \bar{t}_{yy}y + \bar{t}_y) \tag{4}$$

can fix this. The result is a better correspondence between ideal boxes and image boxes as shown in Figure 4c.

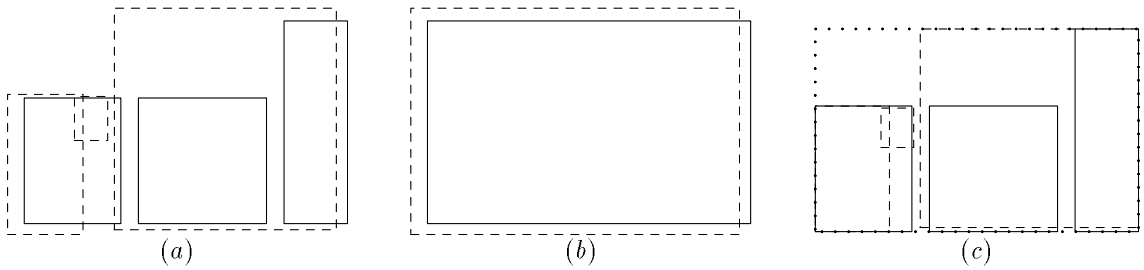


Figure 4: (a) Transformed image boxes (dashed lines) superimposed on the corresponding ideal boxes (solid lines); (b) the word box (solid lines) and the bounding box of the transformed image boxes (dashed lines); (c) The boxes from (a) with the image boxes transformed so that the bounding boxes from (b) coincide to form the dotted box.

The remaining task is to use this correspondence to map image boxes and the associated artwork to the ideal boxes. If this cannot be done by comparing the observed images against the expected character shapes as Kanungo suggests, Algorithm 1 shows how to do it by using an approximate containment test like (3), but with a tighter tolerance. We omit the details in Step 3 where run-together character images get cut up since this cannot be done reliably without knowledge of the character shapes. See [3] for an example of an application where Algorithm 1 was used successfully.

5 Results

The algorithm was implemented in C++ and tested on 28 pages from six different documents. Each page was printed at 600 dots per inch, photocopied once, and scanned or faxed at various resolutions. Table 1 gives basic information about the documents and the scanned images. The

Algorithm 1 How to use an approximate containment test to take the character images assigned to a word and assign them to ideal boxes within the word, cutting up images if necessary.

1. Test each transformed image box for approximate containment in each of the word's ideal boxes and assign the image boxes accordingly. If a transformed image box is approximately contained in more than one ideal box, choose the one where d_f is smallest.

2. For each unassigned image box B, find the set S_B of ideal boxes that intersect the transformed version of B. Label B as a conglomeration of material from members of S_B.

3. If desired, try to cut up each image for box B from Step 2 cookie-cutter fashion, and assign the pieces to the appropriate ideal boxes.

4. Use the character identity and font labels from each ideal box to give ground truth labels for the images assigned that ideal box.

original document descriptions consisted of LaTeX output that was post processed to produce ASCII files listing character identities and bounding boxes for use by the program. This ASCII format was not specific to TeX or LaTeX and could have been generated from a PostScript document description if a suitable conversion program were available.

Document				Average characters or components			
Id	Font	Language	Pages	Truth	200×100	200dpi	400dpi
A	cmr 10pt	English	5	1715	2100	1819	1838
B	Times 10pt	English	6	2371	2655	2453	2519
C	cmr 12pt	English	5	1272	1839	1423	1379
E	cmr 10pt	English	4	1879	2533	2026	2016
G	cmr 11pt	German	4	1643	2257	1831	1821
S	cmr 11pt	Spanish	4	1453	1916	1564	1540

Table 1: Statistics about the test pages and the character images extracted by page layout analysis. Documents A–C are preprints of journal articles and documents E–G are software manuals. The "Truth" column lists the average characters per page from the original document description and the last three columns count connected components in the scanned images.

In order to test the optimization procedures as thoroughly as possible, we start with a simple estimate for the transformation that matches the image to the document description, and then use all available means to reduce the mismatch. The goal is to get the mismatch below a critical value based on half the width of a typical character as suggested in Section 2.2.

The simple estimate for the transformation differs from Kanungo's approach due to a crude attempt to cope with bad feature points from speckles and miscellaneous junk in the page images. We find the four feature points illustrated in Figure 1 for both the ideal boxes and the image boxes, then try to transform any three of the image box features into the corresponding ideal box features. This gives four possible transformations from which we can select the one closest to the identity transform. If none of these appear reasonable, we try again using additional constraints with pairs of feature points instead of triples. An additional heuristic attempts to cope with fax header lines that appear in the images but not in the document description.

Once the initial transformation is chosen, we can pick a subset of the ideal boxes as explained in Section 3.1 and then use the Nelder-Mead algorithm to minimize the mismatch as explained in Section 2.2. If the mismatch remains above the critical value, a second round of minimization uses the same starting point, but bases the mismatch on all the ideal boxes instead of just a subset of them. If this fails to reach the critical mismatch value, it may be that certain ideal boxes cannot be matched.[1] These ideal boxes can be found by examining the last function evaluation in Round 2, and finding the ideal boxes that contribute the most to the mismatch. For instance one 200×100 dpi page image from Document A had mismatch contributions of 15.2 and 14.9 for the 85th and 84th ideal boxes, while no other ideal box had a mismatch contribution of more than 3.9. Round 3 consists of dropping up to three such high-contributing boxes from the mismatch function and repeating the minimization process.

Figure 5 shows how this three-round minimization process performs on the test pages. Since the mismatch function is the L_4 norm of the contributions for the ideal boxes, the mismatch values used in the figure are normalized by dividing by the fourth root of the number of ideal boxes. This means that the coordinates in the figure are essentially in pixel units, so the higher "after minimization" values for the 400 dpi test pages just mean that the actual distances involved in the mismatch computation are staying fairly constant.

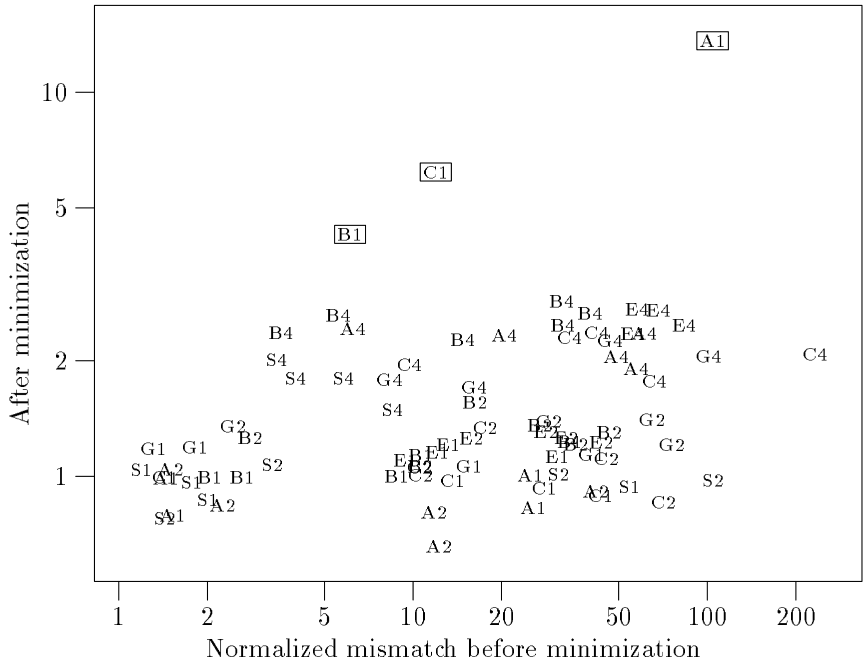


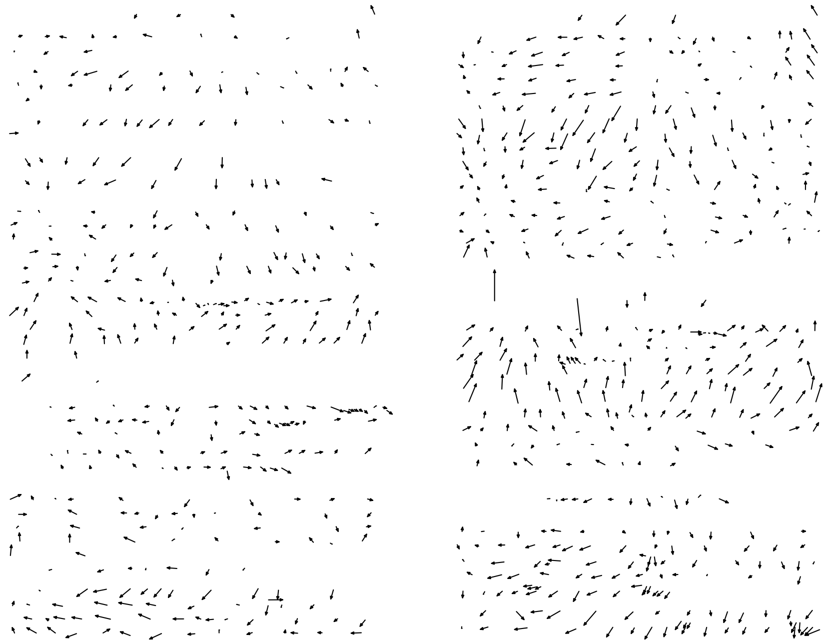
Figure 5: A scatter plot of the normalized mismatch before and after the three-round minimization process for pages from the test documents. Each label gives a document id from Table 1 followed by a digit that gives the resolution: 1 means 200×100 dpi; 2 means 200 dpi; 4 means 400 dpi. Boxes identify pages where the critical mismatch value could not be achieved. All data is for Nelder-Mead minimization.

The three labels above the labels for the 400 dpi test pages are for 200×100 dpi test pages where Nelder-Mead minimization failed to reach the critical mismatch value. Since all other test pages led to less than critical mismatch values, the 200×100 dpi resolution images appear to be the primary trouble spot for Nelder-Mead. This is relatively encouraging in view of the high mismatch values before minimization. Except for the small group of 100×200 and 200 dpi labels at horizontal positions < 3.5 and the small group of 400 dpi labels at horizontal positions < 7, it was apparently not possible to find three appropriate feature points on which to base the initial transformation. Note that minimization often reduced the normalized mismatch by a factor of two even when the initial mismatch value was good enough to suggest that the feature points were appropriate.

If the mismatch values after minimization scale with resolution, is there some underlying cause?

[1]This can happen when character images merge with large connected components such as rule lines in a table.

We test this by viewing the mismatch graphically using the transformation from (4) for each word. Figure 6 shows how the center of each word in the 200 dpi scanned image gets shifted for the two pages from Document A; i.e., it displays the additional transformations needed after the best affine transformation has been applied. These transformations are complicated but they do not look like random noise. If the considerable similarities between parts a and b of the figure are due to the fact that they both originated from the same copier and fax machine, it may be possible to measure the nonlinearities and compensate for them in future experiments as Kanungo suggests [4]. On the other hand, Section 4.2 explained how to find the transformations (4) without such prior calibration if the mismatch is not too great.



(a) (b)

Figure 6: Displacements for the center of each word's bounding box due to the transformation (4) when matching 200 dpi images to the original document descriptions for two pages of Document A. The displacement vectors are exaggerated by a factor of 20.

In view of the importance of getting the mismatch below the critical value, it is worth checking if alternative minimization strategies can reduce or eliminate instances of failure to reach the critical value. One approach is to try other initial transformations. For instance, the failures in Figure 5 can be eliminated by starting at the transformation that turned out to be optimal for the previous page of the document in question. Of course it would be better to find an optimization strategy that outperforms Nelder-Mead on the 200×100 dpi pages.

The guaranteed convergence properties of Torczon's algorithm make it a prime candidate for these difficult, low-resolution test pages. Figure 7 compares the effectiveness of Nelder-Mead minimization with three versions of Torczon's algorithm. The three versions differ in the setting of a parameter that controls the number of function evaluations per iteration of the optimization routine. Torczon refers to this parameter as "the number of search directions." The minimum allowable value of 12 gives rise to the solid round dots in Figure 7; the small open circles are for 24 search directions; and the large open circles are for 96 search directions. The Nelder-Mead data indicated by + signs in the figure do not show a dramatically different distribution.

Figure 7 shows that the mismatch after minimization was always either less 1.5 or more than 4. This separates the data points into those where the mismatch was successfully reduced to the

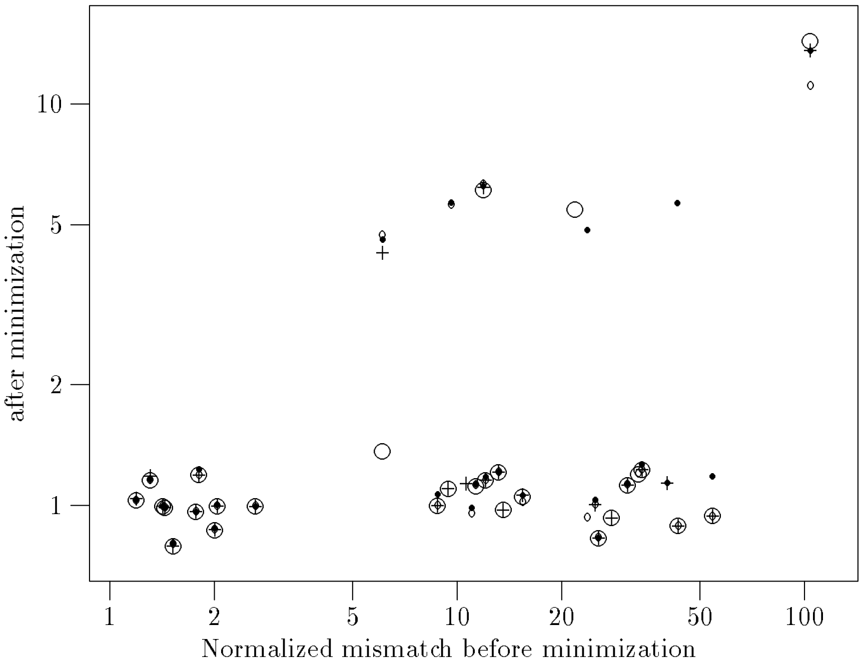


Figure 7: A scatter plot of the normalized mismatch before and after the three-round minimization process for 200×100 dpi pages from the test documents. The + signs are for Nelder-Mead minimization, and the dots, small circles, and large circles are for Torczon's algorithm with 12, 24, and 96 search directions, respectively.

critical value, and those where the minimization process should be considered unsuccessful. Since the most notable feature of this latter group is that dots (Torczon's with 12 search directions) are overrepresented, we tentatively conclude that it is probably best to use more than 12 search directions.

Table 2 shows how Torczon's algorithm compares with Nelder-Mead in terms of the number of function evaluations needed to reach the minimum. The "Overall" row shows that the overall average number of function evaluations is much less for Nelder-Mead. Torczon's algorithm performs best with 24 search directions, but even in that case, Nelder-Mead does fewer than half as many function evaluations. It also helps to increase the resolution. The overall averages are reduced at 200 and 400 dpi because it is seldom necessary to resort to more than one round of minimization.

Scenario	200 × 100 dpi				200 dpi	400 dpi
	Torczon12	Torczon24	Torczon96	NM	NM	NM
Round 1 works	1596	899	2748	307	335	312
Round 2 works	4706	2566	6302	1229	1226	1366
Round 3 works	9525	2673	8853	1918	–	–
Round 2 fails	4485	2590	9134	1648	–	–
Round 3 fails	5361	4149	12405	–	–	–
Overall	2924	1501	4457	657	399	387

Table 2: Average number of times the mismatch function had to be evaluated to process a test page broken down by resolution, optimization strategy, number of rounds of optimization needed, and success in reducing the mismatch below the critical value.

Table 2 may be a little misleading because it weights iterations in Round 1 just like those in Rounds 2 and 3 even though the later rounds make the mismatch function harder to evaluate by basing it on all or almost all the ideal boxes instead of using just 15% of them. Hence, the actual run times in Table 3 show an even bigger speed up at the higher resolutions. Torczon's algorithm still takes more than twice as long, but this could change if parallelism were enabled. (The test machine was an SGI Challenge XL with 12 MIPS R4400 processors running at 150Mhz, but all tests were done on a single processor).

	200×100 dpi				200 dpi	400 dpi
Scenario	Torczon12	Torczon24	Torczon96	NM	NM	NM
Round 1 works	57.6	34.0	96.2	12.5	12.8	13.2
Round 2 works	285.9	169.6	631.7	128.1	77.0	89.9
Round 3 works	833.6	251.5	803.4	158.9	–	–
Round 2 fails	275.3	203.6	568.4	124.5	–	–
Round 3 fails	321.8	358.9	1145.2	–	–	–
Overall	169.5	93.9	281.4	44.3	17.3	18.7

Table 3: Average run time in seconds per test page as a function of resolution, optimization strategy, number of rounds of optimization needed, and success in reducing the mismatch below the critical value.

What about the accuracy of the ground truth produced by the algorithm? This was not tested systematically since the main thrust of this paper is finding the best transformation, and Kanungo has already shown that reliable ground truth can be obtained by template matching against the expected character shapes, once the transformation is known. The test implementation was programmed to reject questionable words rather than risk producing bad output. No errors and very few rejections were found when spot checking the results by hand.

6 Conclusion

When generating ground truth by matching a page image against the original document description, it is of primary importance to find a geometric transformation that maps coordinates appropriately. We have seen how standard optimization techniques can improve the accuracy of such a transformation, even if the initial approximation is way off. In those few cases where optimization does not yield an appropriate transformation, the mismatch function reveals the problem, and restarting the optimization with a different starting point can solve the problem.

A number of important questions remain to be more fully addressed in future work. Does searching for an affine transformation (1) provide the right number of degrees of freedom? Three degrees of freedom are sufficient to determine how a piece of paper is positioned on a scanner, yet affine transformations provide six and Figure 6 suggests that it would help to allow much more complicated transformations. Affine transformations were chosen as a compromise because direct search optimization algorithms have a reputation for behaving poorly when the dimensionality of the search space is too high.

Another question is what optimization algorithm is best. Since Torczon's algorithm is designed to run in parallel, its run times could be significantly better than Nelder-Mead if parallelism is enabled. On the other hand, the mismatch function itself could be parallelized since each ideal box can be considered separately. The tests reported in Section 5 are less than definitive, but the difficulties that occasionally cause Nelder-Mead to fail to find the desired minimum do not seem to be amenable to Torczon's superior convergence properties. The mismatch function could well have undesired local minima.

A possible alternative to direct search methods such as Nelder-Mead and Torczon's algorithm is to try to use differential semblance optimization to construct a smoother or more continuous function to minimize [11, 2].

It also seems promising to try a real segmentation algorithm in place of just connected component analysis. The third round of optimization used in the trials in Section 5 was specifically designed to cope with missed segmentations, but it would undoubtedly be better to improve the segmentation.

Finally, it would help to be able to extract symbol identities and bounding boxes from a PostScript or PDF file, instead of depending on TEX or LATEX output. This would make it easier to generate a wide range of input for the ground truthing process and it would probably allow for more accurate bounding boxes.

7 Acknowledgement

Virginia Torczon and David Serafini provided an implementation of Torczon's algorithm and adjusted it to be more easily called as a subroutine. Margaret Wright's careful implementation of the Nelder-Mead algorithm was also very helpful.

Atts.
References

References

[1] Su Chen, Robert M. Haralick, and Ihsin T. Phillips. Perfect document layout ground truth generation using DVI files and simultaneous text word detection from document images. In *Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, April 1995.

[2] Mark S. Gockenbach. An abstract analysis of differential semblance optimization. Technical Report TR94–18, Dept. of Computational and Applied Mathematics, Rice University, 1994.

[3] John D. Hobby and Tin K. Ho. Enhancing degraded document images via bitmap clustering and averaging. Submitted to ICDAR'97.

[4] Tapas Kanungo. *Validation of Document Degredation Models*. PhD thesis, University of Washington, 1995.

[5] Donald E. Knuth. *The TEXbook*. Addison Wesley, Reading, Massachusetts, 1986. Volume A of *Computers and Typesetting*.

[6] Donald E. Knuth. TEXware. Technical Report CS-TR-86-1097, Dept. of Computer Science, Stanford University, April 1986.

[7] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[8] Ihsin T. Phillips, Su Chen, J. Ha, and Robert M. Haralick. English document database design and implementation methodology. In *Proceedings of the Second Annual Symposium on Document Analysis and Information Retrieval*, pages 65–104, April 1993.

[9] Ihsin T. Phillips, J. Ha, and Robert M. Haralick. Implementation mehtodology and error analysis for the university of washington english document image database-i. In *Proceedings of the SPIE*, volume 2103, pages 155–173, 1994.

[10] R. P. Rogers, Ihsin T. Phillips, and Robert M. Haralick. Semiautomatic production of highly accurate word bounding box ground truth. In *International Association for Pattern Recognition Workshop on Document Analysis Systems*, pages 375–387, October 1996.

[11] William W. Symes. Velocity inversion: a case study in infinite-dimensional optimization. *Mathematical Programming*, 48:71–102, March 1990.

[12] Virginia Torczon. PDS: Direct search methods for unconstrained optimization on either sequential or parallel machines. Technical Report CRPC-TR92206, Rice University Center for Rearch on Parallel Computation, 1992.

[13] Margaret H. Wright. Direct search methods: Once scorned, now respectable. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis)*, pages 191–208, Harlow, United Kingdom, 1995. Addison Wesley Longman.

Matching Document Images with Ground Truth

John D. Hobby

Bell Laboratories
Murray Hill, NJ 07974

ABSTRACT

Since optical character recognition systems often require very large amounts of training data for optimum performance, it is important to automate the process of finding ground truth character identities for document images. This is done by finding a transformation that matches a scanned image to the machine-readable document description that was used to print the original. Rather than depend on finding feature points, a more robust procedure is to follow up by using an optimization algorithm to refine the transformation. The function to optimize can be based on the character bounding boxes—it is not necessary to have access to the actual character shapes used when printing the original.