Diploma Thesis

Document Layout Analysis

Diplomarbeit im Fach Informatik

Image Understanding and Pattern Recognition Group
Deutsches Forschungszentrum für Künstliche Intelligenz

Fachbereich Informatik
Technische Universität Kaiserslautern
Prof. Dr. Thomas Breuel

vorgelegt von:

Joost van Beusekom
Matrikelnummer: 34 33 90

Gutachter:

Prof. Dr. Thomas Breuel

Betreuer:

Daniel Keysers
Faisal Shafait

Contents

Chapter 1

Introduction

For many years people claim that, after thousands of years of predominance of paper, it will disappear as medium to save and transport information. Although many products have been developed to reduce the use of paper, for the moment it is not predictable if paper will ever disappear as information storage. As nevertheless more and more processes of everyday life are done electronically, there is a strong need to be able to convert printed documents into electronic ones. One step to do so is to use optical character recognition (OCR).

One important step in OCR systems is the manipulation of the document layout. Although the text contains most of the information of a document, the layout also has a certain importance. Imagine for example how an OCR software should try to convert a scanned image to a word processor conform document without considering the layout. No proper reconstruction of the document can be done.

Or, if a three column newspaper should be displayed on a PDA display, the document has to be "serialised" in order that the images stay on the right logical position (near the paragraph referring to them) or that the order of the paragraphs is conserved. In a first step this requires to know how the layout originally looked like.

Another example is layout based document image retrieval: as many people recognise documents in a first step by the layout, it would also be favourable to have some possibility to search for documents having a layout similar to a given one. Therefore it is necessary to be able to compare layouts. Considering these applications, it becomes important to take care of the problem of document layout analysis.

In layout analysis there are a lot of different areas of research. A good overview is given by Cattoni et al. in [7]. One major part of the publications are algorithms used to segment a page into homogeneous areas, which is called page segmentation. Another part does logical layout analysis and tries to label the different blocks according to their function in the text (e.g. author, title, abstract, footnote, etc.). On the domain of benchmarking for layout analysis algorithms also a bunch of work has been done. Benchmarking tries to evaluate how good a given algorithm works, e.g. how close the automatically created layout is to the ground truth, normally produced by a human.

Many methods and algorithms exist in layout analysis for many different applications. Despite this collection of methods, there are only few methods for

distance measuring for layouts. Distance measures can be used for several applications, e.g. document image retrieval by layout analysis, page segmentation by example and benchmarking for page segmentation algorithms. This is the reason why in this diploma thesis methods for distance measuring have been analysed. This will be discussed in the first part of this thesis.

In the second part, a tool is presented to visualise the results of layout analysis algorithms. This tool can also be used to convert segmentations of pages to HTML tables, which are a widely spread format that allows simple visualisation on the one hand, and further processing of the segmentation data on the other hand.

Chapter 2

Distance Measures for Document Layouts

2.1 Introduction

The importance of layout analysis having been described in Chapter 1, it also is important to be able to compare document layouts. This comparison can be done using a layout distance measure. A distance measure for layouts is for example used to benchmark layout analysis algorithms: the output of an algorithm has to be compared to ground truth in order to rate it. Another application is layout based document retrieval: in this case a query layout is compared to a database of known layouts in order to find layouts that are similar, e.g. layouts from the same journal or magazine. The main application for the distance measure presented here will be layout based document image retrieval. A third application could be the combination of different layout analysis algorithms: one approach would be to take the output of several known layout analysis algorithms and to compute the distance for these outputs to a database of known "good" layouts. Then the output with the smallest distance is taken as the result of the layout analysis process. The hope is that by this process the specific errors that each of the known algorithms has, will have less effect on the final resulting layout. Another approach could be to take one layout analysis algorithm, to compute a possible layout in the first step, then compare this layout with the database and to transfer the layout of the best match to the new document. This kind of application is also known as layout analysis by example.

There are two categories of layout information: the geometric layout information and the logical layout information. The geometric layout information consists of data describing how the layout is divided into different homogeneous regions. In our case these regions are rectangular blocks that are represented by two points, the lower left and upper right corner coordinates. An example of a document and its geometric layout information can be found in Figure 2.1 and in Table 2.1.

Logical layout information goes one step further and classifies the blocks into different categories: e.g. title, abstract and author. In this diploma thesis we focus on the geometric layout information and do not discuss the logical layout

information.

In this chapter two ideas will be analysed: one distance measure, proposed by Liang et al. in [18], that was initially designed for layout analysis algorithm benchmarking and a distance measure based on a block distance for blocks for two layouts. This method was initially used to compare the output of a segmentation algorithm to the ground truth for a given document image. Based on block information, it computes the different segmentation errors by comparing the overlapping area of the blocks.

The second method uses various distance measures to compute the distance between two blocks and three matching functions that can be composed to a wide variety of different distance measures. As matching methods the minimum weight edge cover, the assignment problem and the Earth Mover's Distance have been chosen. The block distances presented here are based on geometric features of pairs of blocks as e.g. the difference in width, the distance between their centres and their overlapping area. Different combinations of block distances with matching methods have been tested. This will be discussed in the second part of this chapter. An example for one possible combination could be the difference in width of two blocks as block distance, combined with the assignment problem for the matching step.

Block number	X_{low}	Y_{low}	X_{high}	Y_{high}
1	184	2923	1403	2989
2	182	2716	1406	2853
3	181	1538	1520	2439
4	185	1334	484	1366
5	183	1201	744	1241
6	184	1052	1521	1139
7	184	810	605	856
8	1053	805	1516	856
9	847	840	855	866
10	1793	652	1804	669

Table 2.1: The coordinates of the corner points of the blocks of the geometric layout information as shown in Figure 2.1.

2.2 Related Work

Most of the work in the field of document image retrieval uses specific features computed on the document image, e.g. font information, connected components, and texture. An overview of the publications on this subject can be found in [9]. Shin, Doerman and Rosenfeld present in [28] a method using structure-based features for document classification and retrieval. Eglin and Bres [10] present a document similarity that in a first step labels the different regions (e.g. text, image, heading) in the document and then compares these labelled regions to the regions of another document.

We disregard the image as information and use only block information in order to analyse to what extent the geometric layout information alone can be used for document image retrieval. To do so, methods are needed that only

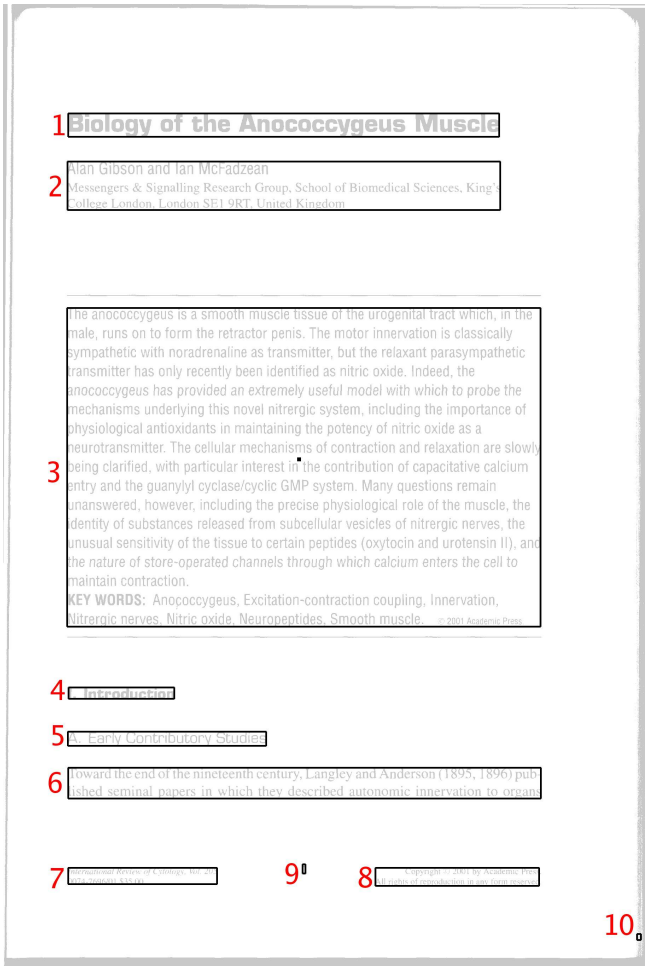


Figure 2.1: Example of a page with its geometric layout information. The original page is painted in grey, the geometric layout information blocks in black. The black point near the centre of the abstract represents the centre of gravity of all the blocks.

use the geometric layout information to measure the distance. However, the above-mentioned features are likely to aid the retrieval if include in the process described here.

Hu et al. present in [13] a two step method for layout comparison. They use different methods to compute the distance between image rows after a segmentation into a grid of equal-sized cells. Each cell is identified as text cell if at least half of the cell is part of some text block. In the other case it is a white space cell. Document images are then compared using dynamic programming on the row-based representation of the documents. In [12] the use of clustering and a hidden Markov model for learning of prototypes is discussed in more detail. The test data used comprises five classes (1-column and 2-column letter, 1-column and 2-column journal, magazine) and an average error rate of 21.4% is reported. Unfortunately, the data used are not available, so a direct comparison with the results is not possible.

Benchmarking methods for layout analysis algorithms have to compare the output of the algorithms to the ground truth. Because these measuring methods yield a quantitative description of the difference between two layouts, they can be used as a distance measure for the task of document image retrieval.

Mao and Kanungo present in [19] their page segmentation evaluation toolkit in which they use morphological operators to define the sets of missed, merged, split, and falsely detected text lines. The error types are weighted and a total metric is obtained by summing up the error types multiplied by their weight.

Liang et al. propose in [18] a method that uses another approach to find the different errors and then also computes the total error by summing up the different errors multiplied by their weight.

In [31] Yanikoglu and Vincent present a method similar to the ones mentioned above, but instead of working with regions or blocks, the number of errors are counted on the basis of pixels. The total error is then again obtained by summing up the number of errors multiplied by the weighted cost.

2.3 Properties of a Good Distance Measure

The distance measure for layouts for image retrieval should give a small distance when two layouts are similar and a long distance when two layouts are different. As mentioned above, the layout is described as a set of blocks. In order to measure the similarity (or dissimilarity) between two layouts, blocks from one layout have to be compared to blocks from another layout and a definition for similarity has to be defined based on these blocks. The following criteria have been regarded as useful for this measurement:

- Position: If the positions of the blocks of the two layouts are similar then this will be in favour for the analysed layouts.
- Width: If the width of the blocks is the same, the two layouts may have the same column width.
- Area: If the two blocks of different layouts differ in area this might be an indicator that these blocks should not be matched to each other.

Furthermore, it would be useful if the distance measure would be tolerant for some typical errors that occur during layout analysis, namely splitting and

merging errors. It should also allow a few false and missing errors. A merging error occurs when two blocks of the query layout are transformed into one block of the reference layout. A splitting error occurs if the inverse case happens. A false error occurs when one block from the query layout is not matched to any other block of the reference layout. A missing error occurs when a block from the reference layout is not matched to a block of the query layout. A last type of errors, spurious errors, are those errors that do not fit in any of the error categories mentioned above. Examples for these errors can be found in Figure 2.2.

Another important feature for a distance measure used for queries is that it should be fast to compute. The distance measure should be able to compare one query document to a whole database of documents in an "acceptable" amount of time, where "acceptable" may be defined e.g. that the user obtains the results for a query in less than 10 seconds, if run on actual hardware.

2.4 Definitions

In the following sections we use these definitions:

- layout: a layout L is a set of blocks B_i: $L = \{B_1, ..., B_n\}$
- block: a block B is a pair of points P_i: $B = (P_1, P_2)$, the lower left and the upper right corner.
- point: a point P is a defined by a pair of coordinates x, y: $P = (x, y)$

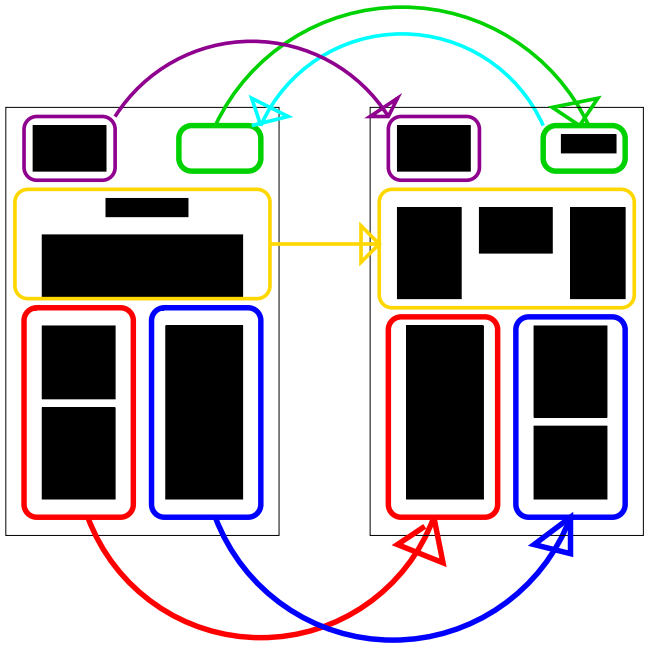


Figure 2.2: The red part is an example of a merging error, the blue of splitting and the cyan one of false and the green one of a missing error; the pink one is a correct match and the yellow ones are spurious errors.

2.5 The Benchmarking Distance Measure

Layout analysis algorithm benchmarking tries to find some objective measure for the performance of a layout analysis algorithm. Benchmarking of layout analysis algorithms is often done by comparing the resulting layout of a layout analysis algorithm to the manually extracted layout of the same page, called ground truth. The idea that lead to the use of this method as distance measure for layouts was, that for benchmarking layout analysis algorithms, the output of these algorithms, namely layouts, have to be compared to the ground truth. So the idea was to take one of these benchmarking methods and try to use its distance measure for our purposes.

One algorithm that was simple to implement and fast to compute was the one presented by Liang in [18]. Because of its small computational effort it would be a good choice for a distance measure for document image retrieval. This method is based on the computation of the overlapping area of the matched blocks. But instead of using this area as distance measure, it uses the relations it gets from this overlapping area analysis to find out what different type of matching errors have been made by the segmentation algorithm.

Six kind of matches have to be distinguished: one-to-one match (no error), one-to-zero match (missing error), zero-to-one match (false error), one-to-many match (splitting error), many-to-one match (merging error) and many-to-many match (spurious error). Then a certain weight is assigned to every kind of mismatch. The final score is obtained by summing up all the mismatches weighted with the specified value and dividing this by the total number of boxes in the two layouts.

The analysis of the overlapping areas, in order to get the number of the different sorts of matches, is done by computing two "area overlap" matrices Σ and T. The entries σ_{ij} and τ_{ij} are defined below:

$$\sigma_{ij} = \frac{Area(B_i \cap B_j)}{Area(B_i)} \tag{2.1}$$

$$\tau_{ij} = \frac{Area(B_i \cap B_j)}{Area(B_j)} \tag{2.2}$$

where $B_i \in L_1$ and $B_j in L_2$ are two blocks from the two layouts we want to compare.

After having computed these values for every i and j, the different kind of errors can be extracted by the following rules:

- if $\sigma_{ij} \simeq 1$ and $\tau_{ij} \simeq 1$: one-to-one match
- if $\sigma_{ij} \simeq 0$, $\forall j$: one-to-zero match
- if $\tau_{ij} \simeq 0$, $\forall i$: one-to-zero match
- if $\sigma_{ij} < 1$, $\forall j$ and $\sum_{j=1}^{N} \tau_{ij} \simeq 1$: one-to-zero match
- if $\tau_{ij} < 1$, $\forall i$ and $\sum_{i=1}^{M} \sigma_{ij} \simeq 1$: one-to-zero match
- in all the other cases: many-to-many matches

where M and N are the number of blocks of the two layouts being compared.

In the case of one-to-one match the blocks are correctly matched (the two blocks have nearly perfect overlap). A one-to-zero match occurs when one block has no common area with any other block and so this can be seen as a misdetection. The opposite case, namely the zero-to-one match is called a false alarm. A one-to-many match is called splitting detection and the many-to-one match a merging detection. The many-to-many detections are called spurious detections. The different weights for the types of detections are given in Table 2.2.

The performance measure f (the measure we use as distance measure) is then defined as:

$$f = \frac{\sum_{t \in types} n_t \times w_t}{|P_1| + |P_2|} \tag{2.3}$$

where *types* is the set of different error types, $|P_1|$ and $|P_2|$ are the number of blocks in layouts L_1 and L_2 respectively and n_t is the number of errors of type t, where $t \in types$.

This measure f is then considered as a distance measure between two layouts.

Type	Correct	Merge	Split	Miss	False	Spurious
Weight	0.0	0.5	0.5	1.0	1.0	1.0

Table 2.2: The error-weights for the different kinds of matchings

2.6 Distance Measures based on a Block Distance and a Solution Method for the Matching Problem

2.6.1 Overview

As the basic entity of a geometric document layout information is a block, it is self-evident that a distance measure for two layouts can based on some distance measure for two blocks in order to obtain a global distance. The idea that emerged from this observation was to try different distance measures for blocks (in the following called "block distance") to get one global distance measure for two layouts.

Using these block distances, for every pair of blocks of the two layouts the distance between these blocks is computed. The problem that arises is to match the blocks from the query layout to the reference layout in order to minimise the total distance obtained by summing up the block distances of the blocks that are matched. This problem is referred to as the "matching problem".

Considering that there are different methods to compute the block distance and also three different methods to solve the matching problem, several combinations with these two steps can be formed. In this work we evaluate the different methods for both steps quantitatively.

An illustration of this two-step distance measure can be found in Figure 2.3 and an example illustrating the matching in Figure 2.5.

In the first part of this chapter we will present a number of block distances. The second part then presents the three different matching methods that have been analysed.

2.6.2 Block Distances

In this part different block distances are presented that can be used to compute the distance between two blocks. This first computation step returns the cost matrices needed for the second step, the matching.

Relative and Absolute Block Positions

The coordinates of the blocks give the exact position of the blocks in the original image. If the same image is translated, e.g. due to a different position of the document on the scanner, the difference for these two layouts, based on absolute block positions will probably high. Therefore it is useful to compute the positions of the blocks relative to some reference point. A possible choice for this reference point is the centre of gravity of the document layout. This centre of gravity is computed by considering pixels that are part of a block as "mass" and then by computing the centre of mass of the layout. Other methods to make the distance measures robust to translations are also possible. Robustness against rotation is not discussed here, as normally a deskewing step is done before the segmentation step.

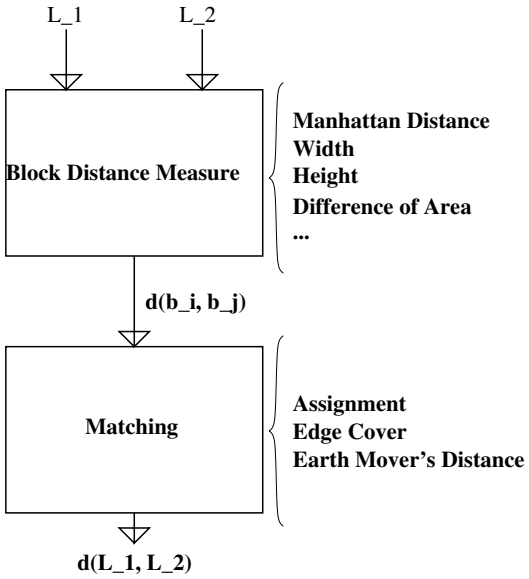


Figure 2.3: An illustration of the distance measure construction kit. As input it gets two layouts, then it computes all the possible distances between the blocks of these layouts and in the last step the matching is done in order to obtain the total distance.

Manhattan Distance of Corner Points

The idea that led to this distance measure was to create a distance measure that implements some simple heuristics for a distance measure between two blocks. As stated above, the criteria for similarity between blocks are similar positions, similar widths and similar areas of the blocks. In order to include these features into one distance measure, we opted for the sum of the Manhattan distances of the corner points of the bounding box as a possible distance measure. Mathematically speaking this means:

Let $B_i = (P_k, P_l)$ and $B_m = (P_n, P_p)$ two blocks of the layouts L_a and L_b respectively. Then the block distance D_{mh} is obtained by summing up the Manhattan distances of the corner points of block B_i and B_m.

$$D_{mh}(B_i, B_m) = d_{mh}(P_k, P_n) + d_{mh}(P_l, P_p) \tag{2.4}$$

where

$$d_{mh}(P_a, P_b) = |x_a - x_b| + |y_a - y_b| \tag{2.5}$$

is the Manhattan distance between two points.

As in this distance position and also area take influence, this might be a reasonable block distance measure to use.

Overlapping Area

This method computes the distance between two boxes by their overlapping area. The overlapping area is defined as the number of pixels that belong to the two boxes being compared and that have the same global coordinates on the page.

For every pair of boxes (B_i, B_k), where $B_i \in L_a$ and $B_k \in L_b$ the following distance is computed:

$$D_{ov}(B_i, B_k) = 1 - \frac{2 \times Ov(B_i, B_k)}{Area(B_i) + Area(B_k)} \tag{2.6}$$

where $Area(B_i)$ is the number of pixels (area) of box B_i and $Ov(B_i, B_k)$ is the overlapping area of block B_i and B_k. We then obtain for every pair of blocks a value between 0 and 1 where 0 is a perfect overlap and 1 is no overlap at all.

This block distance also incorporates position as well as area and aspect-ratio into one measure. Only taking the overlapping area as block distance has one major drawback: in case of non-overlap, the distance will be 1, no matter how far the two blocks really are. Therefore we tried to add, instead of using only the overlap, the normalised Manhattan distance of the corner points in case that there would be no overlap. So for a block having no overlap in common with any other block we get a distance between 1 and 2. The Manhattan distance of the corner points can be normalised by dividing the obtained distance by twice the maximal possible Manhattan distance. This maximum possible Manhattan distance is given by the global lowest leftmost point x_{min}, y_{min} and the highest rightmost point x_{max}, y_{max}. For the ease of computation one can also take the sum of the width and the height of the picture as maximal possible Manhattan distance.

Other simple block distances

It is clear that we can construct various other block distances. As they are almost self-explanatory, they are only defined here:

- Difference in width:

$$D_w(B_i, B_k) = |width(B_i) - width(B_k)| \tag{2.7}$$

- Difference in height:

$$D_h(B_i, B_k) = |height(B_i) - height(B_k)| \tag{2.8}$$

- Product of difference in width and difference in height:

$$D_p(B_i, B_k) = D_h(B_i, B_k) \times D_w(B_i, B_k) \tag{2.9}$$

- Distance of block centre:

$$D_{bc}(B_i, B_k) = d_m(centre(B_i), centre(B_k)) \tag{2.10}$$

It is important to see that these block distances presented here only constitute different choices we can make. It is not said that all these block distances are meaningful or will give good results. The idea is to evaluate how well these simple measurements work and then to try to obtain a better distance measure by combining different simple block distances, each containing a different kind of information, to one new block distance. Examples for these simple block distances can be found in Figure 2.4.

Combination of different block distances

Instead of choosing one block distance to compute the distance between two blocks it might be useful to combine different block distances into one new block distance.

The difference in width, for example, contains information concerning the number of columns we have. But we do not have any information about position. So we might add, for example, the distance of centres of the blocks to the difference in width. As one can see, we obtain a lot of different possibilities bringing some new problems:

- Scale: the different distance measures may have different scales, so they should all be converted to the same scale, e.g. if we want add area to Manhattan distance.

- Addition or Multiplication: to combine these different distances there are different methods, especially adding or multiplying: instead of adding different block distances they also could be multiplied. This is only meaningful in a few cases where one of the block distances has the ability of defining a perfect match, as e.g. for the overlap: if the overlap distance equals 0, than the position and the size and the aspect-ratio of the two blocks is the same, so we have a perfect match. Other ways of combining the simple block distances are also possible.

- Weighting: different block distances could be weighted differently according to their importance regarding our criteria, e.g. if we add the difference in width with the difference in height, it would be useful to give more importance to the difference in width, as for our purpose, column width is more important than paragraph height.

So we get a lot of possible methods where it is interesting to see how each of them will perform.

2.6.3 Matching

In this section the different matching methods will be presented. The matching step matches the blocks from the query layout to the blocks of the reference layout in a way to minimise the total distance. The total distance is obtained by summing up the costs of the matches, which are equal to the block distance between the two blocks participating in the matching. An example is shown in Figure 2.5.

In the literature the matching problems as used here normally are presented using the term "cost" instead of "distance". When we present the general problem we will use "cost", if we are applying the method to our purposes we will use the term "block distance".

Three different matching methods will be discussed:

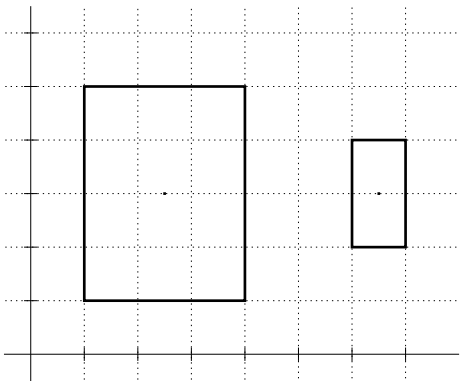


Figure 2.4: Example for simple block distances: the difference in width is 2 units, the difference in height is 2 units, the product of difference in width and difference in height equals 4. The number of overlapping pixels equals zero, so the overlapping area distance will be 1.45 as the maximum Manhattan distance is 11, and the Manhattan distance of the corner points equals 10 units.

- Assignment Problem: each block is matched at most once.
- Minimum Weight Edge Cover Problem: each block is matched at least once.
- Earth Mover's Distance / Transportation Problem: each block is matched partially to at least one other block.

The name of Earth Mover's Distance was chosen by Yossi Rubner in [25] as some CAD programs for road design have a function that allows to compute the optimal way to move earth from roadcuts to roadfills. The Earth Mover's Distance is based on the transportation problem. A very common example to explain the transportation problem is the one of factories producing some goods and consumers consuming these goods. Before the consumers can do so, the goods have to be transported from the factory to the consumers and this transportation has a certain cost. The assignment problem is a special case of the transportation problem: the usual example to explained is uses agents (instead of factories) and tasks (instead of consumers). One agent is assigned to one task. This assignment also has a certain cost, and as for the transportation problem, the optimal solution is wanted. The minimum-weight edge cover problem obtained its name out of graph theory, where the edge cover for a graph denotes a set of edges that connects the vertices. The minimum weight derives from the fact that for this application, the edges have a weight (or cost) and the edges covering the vertices have to be chosen in a way to minimise the total weight.

Matching by Solving the Assignment Problem

As mentioned above, the aim of the matching step is to match blocks from the query layout to the blocks of the reference layout by minimising the total cost. The total cost is the sum of all matches between two blocks multiplied by their cost that in our case will be given by the corresponding block distance.

For the "Assignment Problem" we allow each block to be matched at most once and every block that can be matched should be matched. So we have exactly $min(|L_1|, |L_2|)$ matches. It may happen, that, if the number of blocks of the two layouts differ, some blocks are not matched. These will be called "unmatched" blocks.

As the assignment problem is a bipartite graph matching problem, it can be defined the following way (definitions are taken from [22]):

Let $G = (V, E)$ be a graph consisting of a finite set V of vertices and a finite set of edges E. A bipartite graph G is a graph whose vertex set $V(G)$ can be partitioned into two non-empty subsets X and Y and where for every pair of vertices of one set, no common edge exists.

Now we consider our two layouts L_1 and L_2 as part of a bipartite graph in the way that $V = (L_1 \cup L_2)$.

For every possible pair of blocks $B_m \in L_1, B_n \in L_2$ we assign costs for an eventual edge (B_m, B_n) given by a block distance $d_{gd}(B_m, B_n)$. We now want to determine edges between these two subsets such that the number of edges is equal to the minimum of the size of L_1 and L_2 and the total cost is minimal.

This problem can be solved by the "Hungarian Algorithm" (also called "Munkres' Algorithm"). The assignment problem can be described by a matrix,

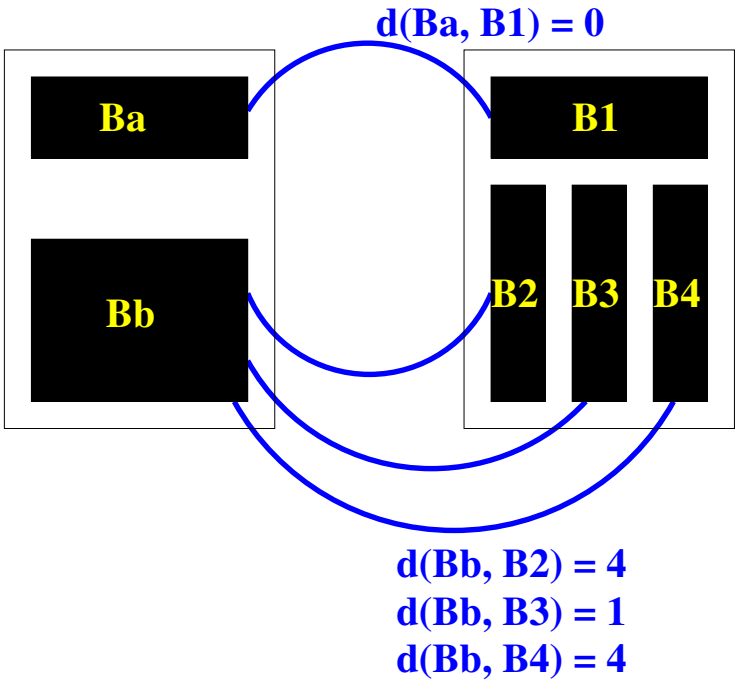


Figure 2.5: Example of matching blocks: in this example every block of the query layout (left) is matched to a block of the reference layout (right). The total costs are given by $0 + 4 + 1 + 4 = 9$. The block distance used is the Manhattan distance of the block centres. This is a minimum weight edge cover solution. The assignment problem solution would match block Bb to B3, B2 and B4 would remain unmatched.

where the entries m_{ij} are equal to the block distance between block i and block j. An example can be found in Table 2.3 and 2.6.3. An example for the optimal solution for the given cost matrix can be found in Table 2.6.3. A description and an example for the Hungarian algorithm can be found in Appendix A.1.

	B_1	B_2	B_3
B_a	$d_{gd}(B_1, B_a)$	$d_{gd}(B_2, B_a)$	$d_{gd}(B_3, B_a)$
B_b	$d_{gd}(B_1, B_b)$	$d_{gd}(B_2, B_b)$	$d_{gd}(B_3, B_b)$
B_c	$d_{gd}(B_1, B_c)$	$d_{gd}(B_2, B_c)$	$d_{gd}(B_3, B_c)$

Table 2.3: Symbolic cost matrix. d_{gd} stands for a block distance.

	B_1	B_2	B_3
B_a	1	2	3
B_b	2	4	6
B_c	3	6	9

Cost matrix

	B_1	B_2	B_3
B_a	0	0	1
B_b	0	1	0
B_c	1	0	0

Solution

Table 2.4: An example for an assignment problem and its solution. Each "1" represents an assignment: block B_c is assigned to block B_1, block B_b to block B_2 and block B_a to block B_3. The optimal solution has a total cost of 10.

Handling non quadratic problems In our application of the assignment problem it often happens that the problem is not quadratic. This is the case when the number of blocks in the two layouts differ. After the assignment there are a number of blocks that are not matched. We have to take care about handling these blocks appropriately. Simply ignoring them would give good similarities for layouts consisting of a few blocks only. Penalising them by some value afterwards could be a possibility but also inserting dummy blocks with a certain penalty distance in order to get a quadratic problem could be a solution, which is the approach we followed.

Matching by Solving the Minimum Weight Edge Cover Problem

The minimum weight edge cover problem consists of finding matches for the same problem as the assignment problem, with the difference that every block of L_1 is connected to at least on block of L_2 and vice versa. This problem is solved using the Hungarian algorithm that was used in the preceding section to solve the assignment problem. A description and an example for the algorithm can be found in Appendix A.2. An example for the assignments done by the minimum weight edge cover matching can be found in Table 2.5.

Matching using the Earth Mover's Distance

One other interesting approach we analysed is to use the Earth Mover's Distance (EMD) as matching method. Instead of matching entire blocks, here blocks can be divided into parts (in our case pixels) that are assigned to other blocks.

	B_1	B_2	B_3	B_4
B_a	1	2	3	4
B_b	2	4	6	8
B_c	3	6	9	12

Cost matrix

	B_1	B_2	B_3	B_4
B_a	0	0	1	1
B_b	0	1	0	0
B_c	1	0	0	0

Solution

Table 2.5: An example for minimum weight edge cover based matching. The left table contains the initial problem, the right one the optimal matching with a total cost of 14.

The EMD was used in [26] for image retrieval of colour images. Simply spoken, the idea behind it was to calculate the cost to transform the histogram of one image to the histogram of the other. In our case we want to build the reference layout by moving pixels from blocks of the query layout to blocks of the reference layout. As moving these pixels has a certain cost, a total cost can be computed to convert one layout into another. Blocks as well as histograms are called "signatures" in this concept, which is a more general concept.

Signatures

A signature is defined as a set of feature clusters represented by their mean and by the fraction of pixels ("earth") that belongs to this cluster: signature $S = \{s_j = (m_j, w_j)\}$ where m_j is the mean value of the cluster and w_j the fraction of pixels that belong to cluster j. A page layout can be considered as a signature: the blocks represent the clusters and the fraction is given by the area of the blocks (the number of pixels in that block).

Transportation Problem

The computation of the EMD is based on the solution of the transportation problem. The following example illustrates what the transportation problem is: Imagine there are two factories producing a certain amount of goods each, e.g. Factory 1 (F1) produces 50 units of goods, Factory 2 (F2) produces 30 units of goods. On the other side there are 3 consumers that all want to consume a certain amount of goods. Consumer 1 (C1) needs 25 units, consumer 2 (C2) 45 and consumer 3 (C3) 10 units of goods. In order to get the goods from a factory to a consumer, they have to be transported, and this transportation has a certain cost. For example: the transportation of one unit of goods from F1 to C1 has a cost of 15, from F1 to C2 30, etc. These informations can be summarised in the so called "transportation tableau" that can be seen in Table 2.6.

	C1	C2	C3	Supply
F1	24	30	40	50
F2	30	40	42	50
Demand	25	45	10	

Table 2.6: The transportation tableau

Coming back to the initial idea of "earth moving" the factories will be replaced by the clusters of a signature S_1 and the consumers by the clusters of a second signature S_2. And the costs to move one unit from signature S_1 to signature S_2 is given by a block distance between the clusters of S_1 and S_2. After having found the solution for the transportation problem for two signatures we obtain the total cost of transforming signature S_1 to signature S_2. This cost is divided by the total flow (the total number of transported units) and then defined as Earth Mover's Distance.

The algorithm for solving the transportation problem is described in Appendix A.3. A solution for the cost matrix in Table 2.6 can be found in Table 2.7.

Implementation Details For the assignment problem we chose to make the problem quadratic, in case the number of blocks in the two layouts differs by inserting dummy blocks. These dummy blocks need a certain penalty distance. Depending on this distance the algorithm tends to make different mistakes: if no penalty is assigned, layouts with few blocks will be too good a match. If the penalty value is too high, the matching method is very inflexible if the number of blocks in the two layouts differ (which may happen if e.g. splitting or merging errors occur). Various methods of defining this penalty have been analysed, e.g. the mean block distance value, no penalty at all, half of the maximum block distance value, etc., and we opted for a penalty value that is given by the maximum block distance value that exists between two real blocks.

Normalising the total distance by the number of blocks has also been tried but did not give better results, due to the fact that by dividing, we lose the information about the number of blocks per page.

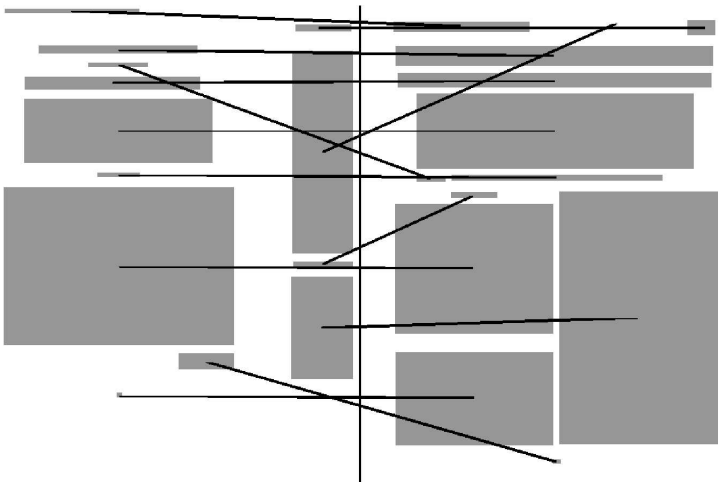
For the minimum weight edge cover method we did not need to specify any parameters.

Examples for the matching result for the assignment and the minimum weight edge cover problem are shown in Figure 2.6. The query document (left) contains blocks on the right side that are not part of the page layout but parts of blocks from the facing page. These artefacts come from the scanning process. The Voronoi layout analysis algorithm was used to extract these layouts. As one can see, the minimum weight edge cover method find a correct match (the same journal), regardless of these "artefacts". The assignment problem based method returns a wrong layout as layouts with approximately the same number of blocks are preferred, due to the penalty value for unmatched blocks.

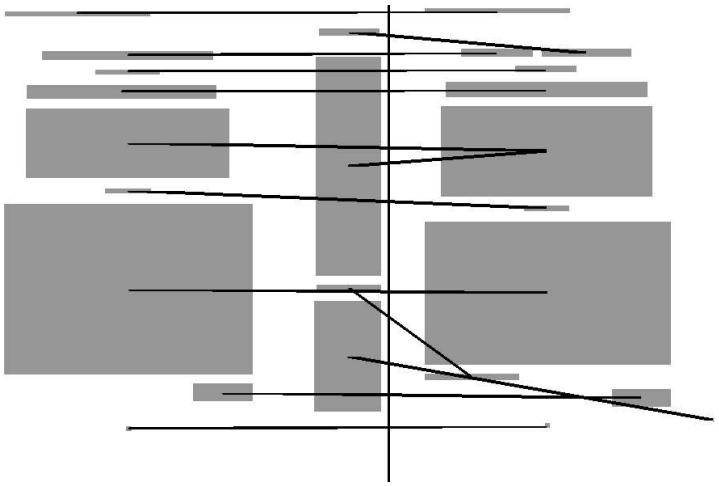
For the Earth Mover's Distance we had to find a solution for the case that two layouts have different number of pixels (only pixels belonging to blocks are considered). As the area plays the role of demand and supply in the transportation problem, the initial transportation problem is unbalanced. So dummy

	C1	C2	C3	Dummy
F1	5	45	0	0
F2	20	0	10	20

Table 2.7: Optimal solution for the transportation problem given in Table 2.6. The total cost C_t is obtained by summing up the product of the allocation times the costs for each cell. $C_t = 5 \times 24 + 45 \times 30 + 20 \times 30 + 10 \times 42 + 20 \times 0 = 2490$



(a) Best match for the left side query layout by assignment problem matching using the overlapping area as block distance.



(b) Best match for the left side query layout by minimum weight edge cover matching using the overlapping area as block distance.

Figure 2.6: Examples for the two matching methods. The lines indicate which blocks are matched to each other. The block distance used in both cases is the overlapping area. The best match for the assignment method finds a wrong layout (not the same journal), whereas the minimum edge cover method finds a correct layout (same journal). The layouts were extracted using the Voronoi layout analysis algorithm.

blocks have to be inserted to solve the transportation problem. These dummy blocks do not have a position nor a size, they simply are pixel "producers" or pixel "consumers", in order to solve the transportation problem.

For the usual application of the transportation problem, this method works fine as there can only be transported as much as there is supplied and needed. The fact that the demand or the supply are too high will have no effect on our total cost.

This is not appropriate for our purpose. We want to take into account all pixels, even if the number of pixels in the blocks of the two layouts may differ. As for the assignment problem, there are at least two possibilities: make the problem a balanced one or penalise the unmatched pixels afterwards. As the second solution needs some intelligent way of setting a penalty value and the first solution gave better results, we opted for the first solution: we normalise each layout to a size of one and each block does not have a fixed size in pixels but only a fraction of pixels of the layout that belong to it.

An example how the pixels from one layout are matched to the other layout for the two methods is shown in Figure 2.7.

2.7 Evaluation of Distances Measures

2.7.1 Evaluation Method

In this section we describe the evaluation method we chose for our distance measures. The problem with evaluating the distance measures is to define what a good result should be like. It should return for a given query layout a reference layout from a database that "looks" similar, but the problem is to provide a ground truth for that notion without too much manual labelling.

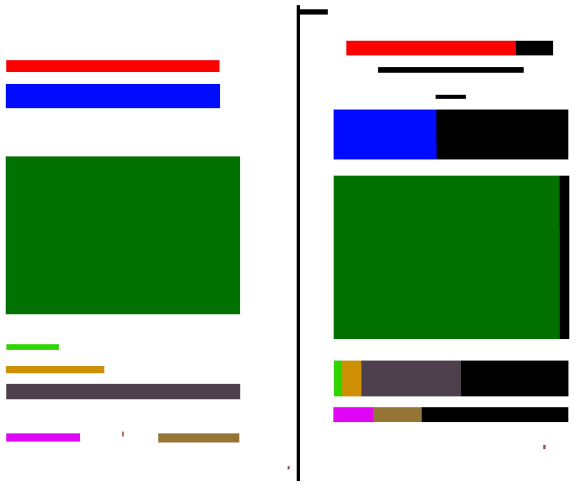
One possibility would be to let a human decide whether the layout found in the database looks similar to the query layout. This idea has several drawbacks, especially the time consuming task of visually comparing the results. Another problem is the lack of objectivity, as every user has a slightly different opinion of what is similar and what not. So the first tests, run on the UW3 Database, only gave very little clues about how good the distance measures performed.

A more objective method for evaluation is needed. The idea was use some document image database that is sorted according to different journals or magazines. As for one specific journal the layout should look very similar for different articles, a query with a document from one given journal should return a layout of the same journal.

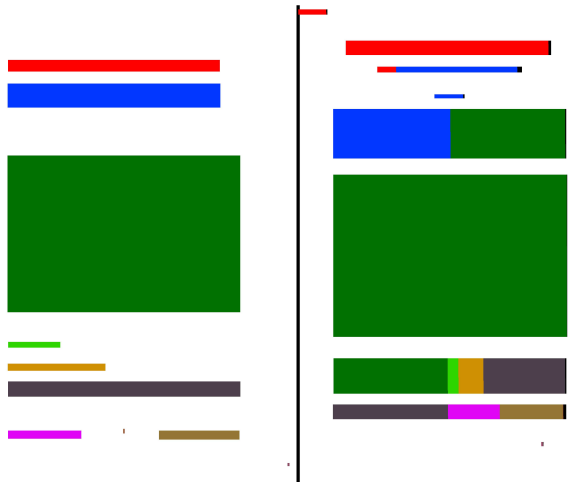
The database we chose for this task is the MARG (Medical Article Records Ground-truth) database. It contains 815 scanned documents of first pages of medical journals, sorted by type (9 different types) and journal (161 different journals). Further details about the MARG database[1] can be found in [11]. The 9 different layout types are shown in Figure 2.8.

As different journals in this database are published by the same publisher and publishers often use similar layouts for their journals, we furthermore manually sorted the MARG database according to the publisher. After sorting we obtained 59 different publishers.

[1] http://marg.nlm.nih.gov/index2.asp



(a) EMD using absolute area of blocks



(b) EMD using normalised area of blocks

Figure 2.7: (a) Result of the EMD without normalisation. (b) Result of the EMD with the areas normalised to one. The left side shows the query image, the right side the best match. The colours indicate the corresponding pixels, where in (a) black is used to indicate unmatched pixels. The block distance used is the sum of the Manhattan Distance of the corner points.

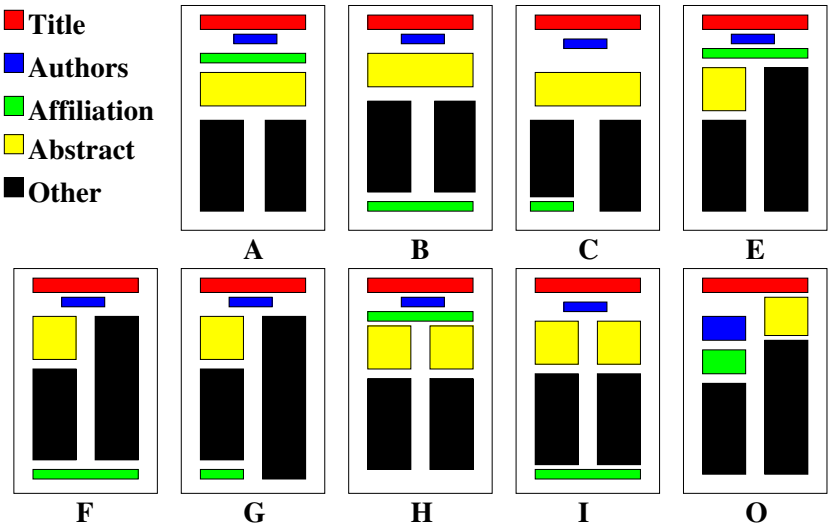


Figure 2.8: The 9 different layout types of the MARG database. Type "0" stands for "Other" and groups different layouts that do not fit into any of the other 8 types.

The testing method used is the leaving-one-out test. As the MARG database has no explicit training or test set, each of the 815 documents is used once as query layout and is then compared to the remaining 814 layouts. The layout with the smallest distance then is the result of the query. If the result of the query is from the same class (journal, publisher or type respectively), the results considered as correct, otherwise it is considered as an error.

As layout information we first used the layout ground truth of the MARG database. However, this ground truth only contains information about four special kinds of blocks, namely "author", "abstract", "affiliation" and "title". After a few tests, it became clear that this partial layout information was not a good basis to test the performance of the distance measures, because many blocks are disregarded. Completely segmented pages are a better choice to run these tests.

We therefore chose to use known layout algorithms to extract the layout. This is also the case which is more relevant to a practical task, because in these it cannot be expected to have access to manually extracted layout information. Although the algorithms will not yield perfect segmentations, they should produce similar errors for similar layouts. The algorithms we used for extracting the layout information are: Voronoi algorithm [16], XY-Cut algorithm [21], Run length smearing algorithm [30], Docstrum algorithm [23], Whitespace algorithm by Baird [2] and Whitespace algorithm by Breuel [4]. An example of a page segmented by all of these algorithms can be found in 2.12.

2.7.2 Layout Analysis Algorithms

In the following, we briefly describe the different layout analysis algorithms used.

XY-Cut

The idea of this algorithm and also its implementation are very simple: the pixels of the image of the document are projected horizontally and vertically. Then we look for the largest possible white gap in the projection and split the image into two sub-images at this gap. We repeat this procedure recursively until a stopping criterion is fulfilled.

We obtain by this method a list of horizontal and vertical splits that segments the whole image into different regions. If we now shrink these regions to the smallest possible rectangle covering all black pixels in this region we obtain the different blocks. Depending on the stop criterion, we obtain lager or smaller blocks.

This method also a few drawbacks. It has problems with separator lines and also with black copy borders that are typical for scanned or copied book pages. In the case of presence of such borders, the algorithm will not cut anywhere because it does not find a gap at all. That is the reason why we first need to remove the black borders from the images of the database before running the XY-Cut algorithm. It also can only segment Manhattan layouts. These are layouts consisting of rectangular blocks only.An example of a page segmented by the XY-Cut method can be found in Figure 3.1 on page 47.

Run Length Smearing Algorithm

This algorithm by [30] is based on the smearing of black pixels in a binarised image. The smearing process smears the black pixels (represented by a 1) over the page in a manner that small white pixels (represented by a 0) are blackened. This smearing is handled by two simple rules:

- Rule 1: 0's are changed to 1's if the number of adjacent 0's is less than or equal to a certain threshold C (if the length of a sequence of 0's is less than or equal to a threshold, the 0's are changed to 1's)
- Rule 2: 1's stay unchanged

Consider the following example, where 0 represents white and 1 black pixels and where the first line represents a line of pixels in its original form and the second line the output of the smearing step of the first line. The smearing threshold C is set to $C = 4$.

```
0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0

1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1
```

First this smearing process is done with the entire document image in horizontal direction with a threshold $C_h = 300$ and a first smeared image is obtained. Then the same is done in vertical direction with threshold $C_v = 500$. These thresholds have been fixed empirically. Then, these two bitmaps are combined with the pixel-wise AND operation. An example can be found in Figure 2.10 This bitmap is then smoothed again by the smearing algorithm with a threshold $C_s = 30$. Now we have the final image as can be seen in Figure 2.10(e).

After this step the bounding boxes have to be extracted, which can be done by a connected component analysis. This step takes as input the image and returns the bounding boxes for each connected component. A connected component is a set of black pixels that are connected to each other. We are considering the upper, the lower, the left and the right pixels for this neighbourhood, a so called 4-neighbourhood (in contrast to the 8-neighbourhood where the diagonal neighbours are considered, too). The bounding box of a connected component is the smallest rectangular box that contains all pixels of the connected component. An example of connected components can be found in Figure 2.9.

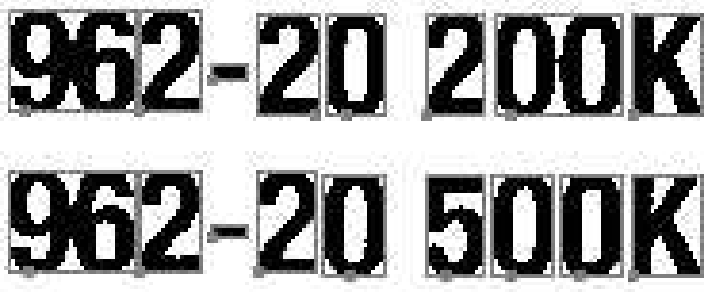


Figure 2.9: An example of connected components. The grey boxes are the bounding boxes of the connected components.

Although this method is quite simple and in some cases performs well, there are these three parameters that depend on the resolution of the image and that have to be adapted correctly. Furthermore it only segments pages having rectangular blocks.

Docstrum

The Docstrum method extracts in the first step different important measurements of the document image. In the second step it groups the different connected components to different regions via these measurements.

The Docstrum method is based on connected components that usually represent individual characters, part of characters and merged characters as well as some punctuation signs. For each of these connected components we compute the k-nearest-neighbour. The k-nearest-neighbour computes the k first connected components that are the closest to the initial connected component, using some block distance. In this case the Euclidean distance is used, and the number k is set to $k = 5$. For each connected component we then get 5 neighbours to which we have a distance d and we can compute an angle θ. So for every pair of connected components i and j we get $D_{ij}(d, \theta)$.

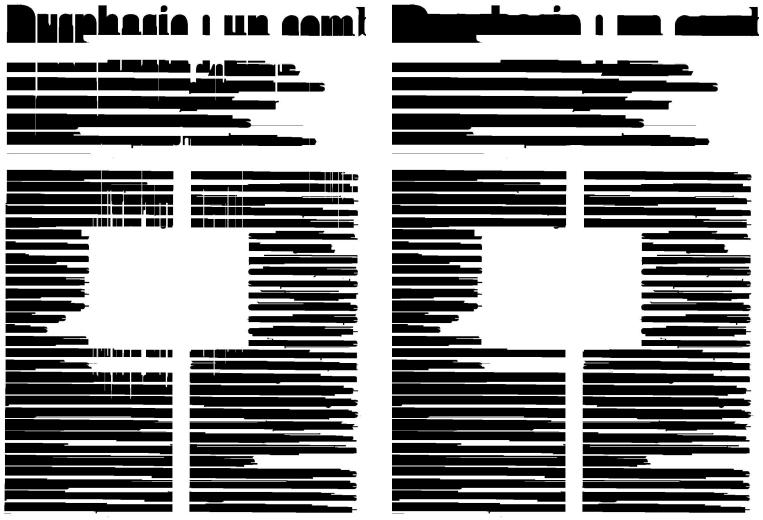
When this data is plotted one can see different clusters and one can extract the different distances that interest us, namely distance to the next character in the same line and distance between lines as well as the skew angle. This is the angle that gives us the page rotation (e.g.: the page was slightly rotated.

Using these distances, merging of connected components to lines by the between-characters spacings can be done. Afterwards these lines can be merged to blocks using the between-lines spacings in order to obtain a fully segmented page.

(a) Original document image

(b) After horizontal smearing

(c) After vertical smearing



(d) After AND

(e) After Smoothing

Figure 2.10: Example of the different smearing processes

Voronoi

The Voronoi document segmentation algorithm uses so called Voronoi Diagrams to segment a page. A Voronoi point diagram is a diagram that splits up a region consisting of points so that for each point, the area next to him is associated with it. Examples of point Voronoi diagrams can be found in Figure 2.11. The lines on the border of a region are called Voronoi edges.

The Voronoi segmentation algorithm uses a so called area Voronoi diagram. Therefore the border points of the connected components are used. For a certain number of these border points the Voronoi diagrams are constructed. Then, all the Voronoi edges of points of the same connected component are deleted. After this step each connected component is surrounded by a box that in general is not rectangular block. Then, in a last step, further Voronoi edges are deleted in order to obtain regions. These regions then have to be fitted into a block as we are only considering Manhattan layouts.

Whitespace Algorithms

The Whitespace algorithm by Baird [2] uses another approach to segment the page: instead of using the printed black area to segment the page, it tries to find white spaces between the columns and the lines in order to segment the document image.

The first step consists of finding maximal white rectangles. These are rectangles that cannot grow in size without adding one or more black pixels. Then the rectangles that were found will be rated according to the area and the aspect-ratio of the rectangle. The N-best rectangles will then be chosen as a partial description of the background. Out of this description the blocks can be extracted.

The main difference between Baird's and Breuel's method is the rectangle finding step. A few parameters also change, but the approach remains the same. Example for whitespace analysis can be found in [5]

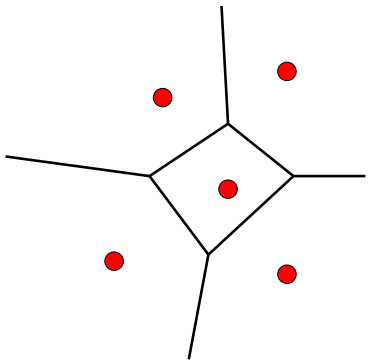
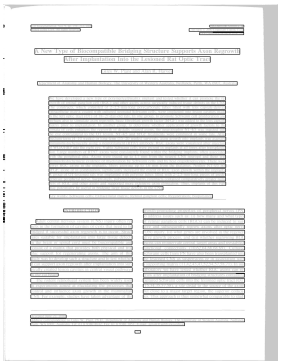


Figure 2.11: Example of a Voronoi point diagram for three points. The red dots are the points for which the Voronoi borders (black lines) are drawn. This example is used to illustrate the idea of the Voronoi diagram. It is not drawn exactly.

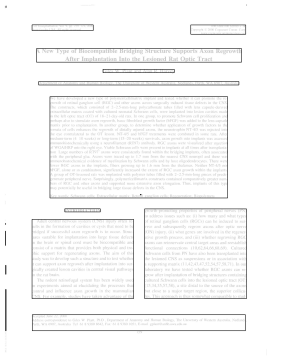
A New Type of Biocompatible Bridging Structure Supports Axon Regrowth After Implantation Into the Lesioned Rat Optic Tract

Giles W. Plant and Alan R. Harvey

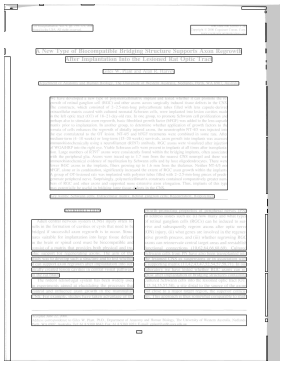
(a) Original document image



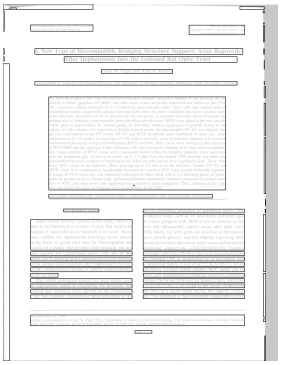
(b) Run-length smearing algorithm



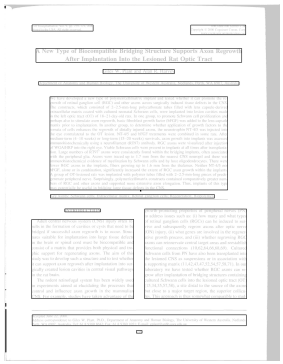
(c) XY-Cut



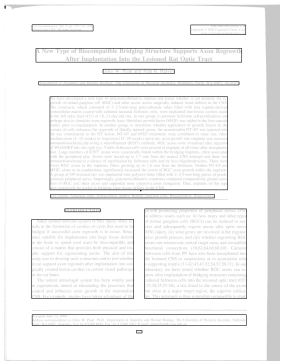
(d) Whitespace (Breuel)



(e) Whitespace (Baird)



(f) Docstrum



(g) Voronoi

Figure 2.12: Example for layout analysis algorithms

2.8 Results

2.8.1 Error Rates

There are three different error rates that were computed using the MARG database in order to evaluate the different distance measures:

- Journal (JOUR): This is the error rate for finding the correct journal for a given query document out of a specific journal. This error rate is the one that should give us, according to the main idea of this evaluation method, a good overview about how good the distance measure works, as we expect to find for a given query document a document of the same journal. There are document images from 159 different journals.
- Type (TYPE): This error rate gives the ratio of misclassifications of the document type. This error rate should be low, as the distance measure should be able to identify the right layout type. Furthermore, only nine different layout types have to be distinguished.
- Publisher (PUB): This error rate gives the ratio of misclassifications of the document publisher, e.g. if the query document is from publisher Elsevier but has been classified as publisher Springer. This error rate is less important because it may be that two journals from the same publisher have different layout, and also the inverse case is quite frequent.

As the images of the MARG database are scanned very precisely, it is not necessary to compute the coordinates of the blocks relative to the centre of gravity of the layout. All results displayed here were gained with absolute coordinates only.

2.8.2 Evaluation of Matching Methods

As mentioned in Section 2.6.3, three different matching methods have been tested: the assignment problem, the minimum weight edge cover problem and the earth mover's distance based on the transportation problem.

These three methods have been tested with different block distances in order to find out whether one method has an overall advantage over the other or if the performance of the matching depends on the block distance, in a way that a good block distance may compensate a bad matching.

As the ground truth contains only incomplete page segmentations, we ran the major part of the tests on the output of different layout analysis algorithms. As [27] stated that the Voronoi layout analysis algorithm is generally a reasonable choice, we chose this algorithm to illustrate our results in this part.

Table 2.8 shows the error rates obtained by applying the three different matching methods. The results are representative for various tests with different layout analysis algorithms that have been performed: in short one can say that the minimum weight edge cover method performs acceptably, whereas the EMD and the assignment method perform worse, with a small advantage for the EMD. Liang et al.'s method, originally used for benchmarking, is not appropriate for our purposes, according to the results obtained. An example for the best match with the benchmarking method can be found in 2.13. As one can see there, the matching is somehow reasonable but there are many errors. One reason

why it performs not that good could be the fact that we implemented the method on our own, so differences may exist between our implementation and the original implementation. Another problem could be that the method to find the matchings does not account for the size of the blocks. A small block that is matched wrongly gets the same penalty as a huge block, although mismatched small blocks do not necessarily mean, that the two layouts are not similar, whereas huge blocks that cannot be matched correctly have a higher impact on the visual difference of the layouts. One other problem are the many parameters that need to be set: the penalty values on the one hand and the threshold that defines the limits the interval of acceptable values for the different steps of the matching computation (e.g. 0 has to be defined as an interval, because perfect overlap is very improbable).

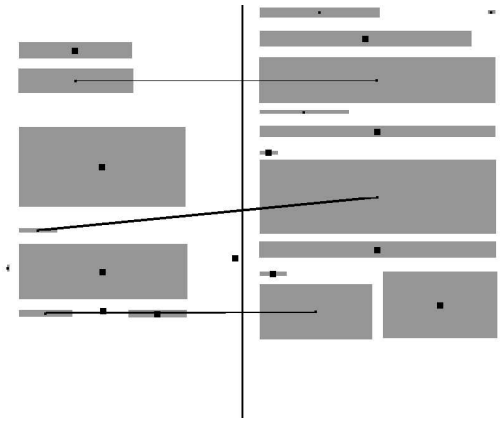
The reason why the assignment problem performs not as good lies in the penalty for unmatched blocks: for unbalanced assignment problems (L_1 and L_2 have different number of blocks) unmatched blocks remain, which are penalised with the maximum occurring block distance value. This implies that layouts with similar number of blocks will be preferred, which is not always a wanted effect. Different other methods for penalising have been tested, without big improvement (the error rate varies, but it stays far away from that of the minimum edge cover): instead of giving a high penalty value for dummy blocks we gave them no penalty at all. Although one might think that this approach is a bad idea, it gave slightly better results. Another idea was instead of penalising the dummy blocks or not penalising them at all, we gave them some neutral cost, e.g. the mean distance of all the possible distances. This solution gave worse results than penalising them. So we conclude that the assignment method has one major drawback, namely the handling of unmatched blocks.

Compare the EMD to the assignment method, we observe that the EMD performs slightly better. If we interpret these results we can say that prohibiting the blocks from splitting up (as does the assignment problem) is not a good idea, as it penalises very much the splitting and merging errors, errors that occur very frequently in document layout analysis. Splitting blocks up into very tiny parts (pixels) is not a good idea either. As one can see in Figure 2.7, pixels from one block may be spread everywhere on the page, a problem that is triggered by the transportation problem, but that is not necessarily wanted for document layout comparison. The minimum weight edge cover method is the most "natural" method for matching layout blocks: merging and splitting are not too expensive, as two or more blocks may be matched to the same block and based on a good block distance blocks are not matched all over the page.

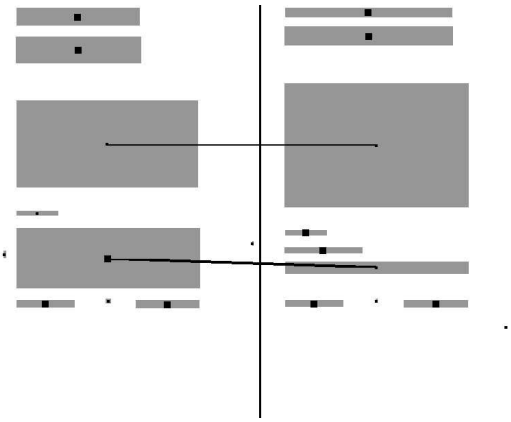
Another important result is the runtime: in Table 2.9 the runtimes for the leaving-one-out test can be found. The block distance used is overlap combined

Distance Measure	JOUR	TYPE	PUB
Edge Cover	32.8	8.2	7.6
Assignment	52.0	22.9	25.4
EMD	52.3	20.2	23.9
Liang et al.	97.2	80.0	93.5

Table 2.8: Comparison of different matching methods (error rates in [%]). The used block distance is the overlapping area.



(a) Best match for Liang et al. method



(b) Best correct match for Liang et al. method

Figure 2.13: Example of matching done by Liang et al. method. The upper figure shows the best match, the lower the best correct match (the best match from the same journal type). Big black rectangles in blocks symbolise many-to-many matches, small rectangles zero-to-one or one-to-zero matches. Lines symbolise one-to-many, many-to-many or one-to-one matches.

with Manhattan distance in case of non overlap. We can see, that the assignment method and the minimum weight edge cover do not differ much. The fact that the minimum edge cover is faster although it uses the assignment method is explained by the size of the problem: for the minimum edge cover the assignment problem is solved for a quadratic size of $min(m, n)$, where m is the number of block in layout L_1 and n the number of blocks in layout L_2. When we use the assignment problem as matching method, we make the problem quadratic to the size of $max(m, n)$. Compared to the Earth Mover's Distance, both are very fast. The 28 minutes that the Earth Mover's Distance need can certainly be improved, but is stay a more complex problem as the assignment problem.

Matching Method	Runtime
Minimum weight edge cover	0m54s
Assignment	1m02s
Earth Mover's Distance	28m31s

Table 2.9: Comparison of the runtimes for the matching methods. The tests were run on an Opteron 2.4GHz.

2.8.3 Evaluation of Block Distances

As shown in the preceding part, the best matching method of the three proposed methods is the minimum weight edge cover matching. In order to test the different block distances we used the minimum weight edge cover together with a few block distances and compared the results. The results were obtained with the blocks extracted by the Voronoi layout analysis algorithm on the MARG database. In Table 2.10 the different error rates for the various block distances can be found.

"Overlap" uses the overlapping area as block distance. So for every pair of blocks we obtain a value between 0 and 1. If two blocks have no common area at all, they will get the distance 1. In that case no conclusions can be made how similar these two blocks are. Therefore we used the block distance "Overlap + Manhattan", that, in case of non-overlap, adds the sum of the Manhattan distances of the corner points divided by the maximal possible distance on the page (sum of length and width of the image) to the 1 of the overlap distance. This way we obtain a value between 0 and 2 for every block and we have some information which block could be better or worse to match to if we have no overlap.

"Manhattan Dist. of Corners" simply sums up the Manhattan distances of the corner points of the two blocks. "Euclidean Dist. of Corners" does the same but instead of the Manhattan distance it uses the Euclidean Distance to measure the distance between two points.

"Manh. Dist. of Block Centres" computes the Manhattan distance of the centre points of two blocks. "Eucl. Dist. of Block Centres" uses the Euclidean distance instead of the Manhattan distance.

"Difference in Width" uses the difference in width of two blocks as block distance, so no explicit position information is used at all.

"Diff. Height × Diff.Width" uses the product of the difference in width and the difference in height of two blocks. Instead of multiplying these two, one can

also sum them up. This is done in "Diff. Height + Diff.Width".

"Difference in Area" uses the square root of the difference of the area of the two blocks.

"Difference in Height" computes the difference of the height of two blocks.

As we can see, the overlapping area as block distance works quite well, compared to the other methods. This comes from the fact that the overlapping area depends on the position and on the size and also on the aspect-ratio of the blocks, so a lot of information is contained within this single measurement. The fact that adding the Manhattan distance of the blocks in case of non-overlap did not improve noticeably the overall performance may come from the observation that the best matches normally are made between blocks that are similar and the rest of the blocks is matched against some other block, although these are not necessarily similar. So it does not make a difference if we match them randomly as done for the first method or if we try to improve the matching by adding some additional feature.

Another conclusion that can be drawn is that the difference in width of two blocks contains much more information for layout comparison than the difference in height of the blocks. This is quite obvious as the column width for all the blocks in a column is the same, but the height of these blocks may vary. In addition, the column width for one journal is typically fixed, so it is a better measure than the difference in height to identify the journal.

Furthermore it can be seen that the Manhattan distance has slightly better results than the Euclidean distance, although the difference is not large.

Comparing "Diff. Height × Diff.Width" with "Diff. Height + Diff.Width", it can be seen that in this case the multiplication is the better operation to combine the two distances. Although the journal error rate is slightly higher, the type and the publisher error rates are lower. Using the sum to combine the two measures will lead to small distances only for blocks with approximately the same length and the same width, whereas the multiplication will lead to small distances if either the difference in length or the difference in height are small. A possible explanation for this behaviour may be, that due to the fixed column width, it happens frequently, for layouts with the same column width, that the total difference in width will become small, although the difference in

Block Distance	JOUR	TYPE	PUB
Overlap + Manhattan	31.2	7.4	7.0
Overlap	32.8	8.2	7.6
Manhattan Dist. of Corners	39.7	11.3	10.5
Euclidean Dist. of Corners	40.7	11.9	11.5
Manh. Dist. of Block Centres	41.6	13.1	13.9
Eucl. Dist. of Block Centres	43.7	14.3	14.8
Difference in Width	47.4	19.4	20.4
Diff. Height + Diff.Width	49.6	17.2	18.4
Diff. Height × Diff.Width	50.7	13.2	14.6
Difference in Area	81.8	54.3	63.3
Difference in Height	88.1	60.1	70.9

Table 2.10: Comparison of the different block distances (error rates in [%]).

height may still be large.

Having a general look at the error rates, even the best of our block distances has an error rate of 31%. This seems to suggest that a lot of improvement should still be possible. However, it is unclear what the class overlap of this task is, i.e. how many journal pages cannot be distinguished by the layout alone.

Recall also that the distance measures use *only* the layout information, i.e. the corner coordinates of the blocks. It is highly likely that better error rates can be achieved when including more information about the blocks, e.g. their texture, the distribution of bounding box sizes, or the output of a text/graphics classifier. Furthermore, the method is able to determine the correct layout type in 92.7% of the cases, which seems a reasonable basis for its use in document image retrieval.

2.8.4 Comparison to the FIRE Results

To compare the results of our document image retrieval method by comparing layouts to FIRE[2] (Flexible Image Retrieval Engine), that originally was developed to do content based images retrieval on pictures, in a first step the original images of the MARG database are used. FIRE computes a different features and then runs the same test as described in Section 2.7.1 (leaving-one-out). The computed features are:

- Image Features: The images are scaled down to the same size and then compared using the Euclidean distance, c.f. [8].
- Tamura Texture Features: Coarseness, contrast and directionality of the images are computed and compared. Further details can be found in [29].

The results are shown in Table 2.11. For the document type one can see, that this method achieves better results as the presented distance measure, keeping in mind that FIRE uses the original more information (original image) as the distance measure (only block information).

To have a comparable results, the same test on images created of the Voronoi segmentation information was run on FIRE. The images contained only black blocks, so that there is no more text and it uses the same information as the distance measure (block information). For this test two other features were computed: Tamura and thumbnails. The weight is 2 for Tamura and 1 for the image features. The error rate can be found in the last line of Table 2.11. It shows that the distance measure presented in this work performs slightly better then the FIRE.

Features	JOUR	TYPES
Only Tamura	24.2	8.2
Only Image Features	34.1	16.3
Tam. & Img Feat. (1:1)	21.8	7.1
Blocks, Tam. & Img Feat. (2:1)	33.0	10.6

Table 2.11: The error rates in [%] of FIRE.

[2] http://www-i6.informatik.rwth-aachen.de/~deselaers/fire.html

2.8.5 Error Types

Having a closer look at the different kinds of errors that the overlapping distance measure produces, we can see in Table 2.12 the different error types together with the number of occurrences of each error. The last column contains the percentage of the error type with respect to the total number of errors. The four error types are the following:

- ¬J ¬T ¬P: neither the journal nor the document type nor the publisher have been recognised (total failure).
- ¬J ¬T P: only the publisher has been recognised.
- ¬J T ¬P: only the type has been recognised.
- ¬J T P: type and publisher have been recognised, only the journal is wrong.

The error where only the journal is wrong is the most frequent. Having a look at an example of this error in Figure 2.14 one can see that this type of error it is comprehensible, as the layouts look even for a human person quite similar. Although only one example is shown, more examples have been looked at and nearly all of them had a high similarity between the query layout and the false match.

By visually controlling errors where neither the right journal, the right type nor the right publisher were found, one can see that these are real failures of the distance measure. An example for such a failure can be seen in Figure 2.15. The upper figure shows the best match found. The lower figure contains the best match for a layout of the same journal as the query layout. This best match layout fits very well, but nevertheless the global best match has a lower distance to the query image. Although the matching is good, it seems as if the splits are more costly than the differing block widths and positions in the global best match. In order to have an idea in what scale the differences of the two matches differ, the measured distances are given here: the upper match has a total distance of 5.72 whereas the lower has a distance of 6.63.

An example for the right publisher but wrong journal and wrong type can be found in Figure 2.17(a). As shown in the example, the two layouts are very similar. Having a closer look to the errors of this type revealed that the database is not error-free. The layouts in the example are of the same type, although the database keeps them as two different types, which is clearly wrong. All 22 errors of this type have been visually checked and in 6 cases, there was a real error. In the other 16 cases, the documents in the database were wrongly classified. An example for one of these errors in the database can be found in Figure 2.16.

Errors where the neither the journal nor the publisher are correct found are rare. In most cases these errors have quite different layouts (12 cases), in some rare cases they result in similar layouts (5 cases). Being difficult to draw a proper conclusion out of the visual control of these errors, it showed that most of the errors leading to different layouts were layouts with a lot of blocks and some segmentations errors. An example for this error type can be seen in Figure 2.17(b).

Having seen these error types, further investigation can to be done: the first thing to try is to reduce the real failures. This could e.g. be done by using

Error Types	Absolute	Relative
¬J ¬T ¬P	39	15.4%
¬J ¬T P	22	8.7%
¬J T ¬P	17	6.7%
¬J T P	175	69.1%

Table 2.12: The different error types together with their absolute and relative number of occurrences

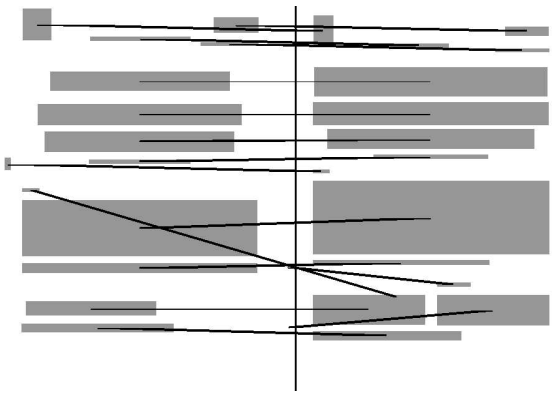
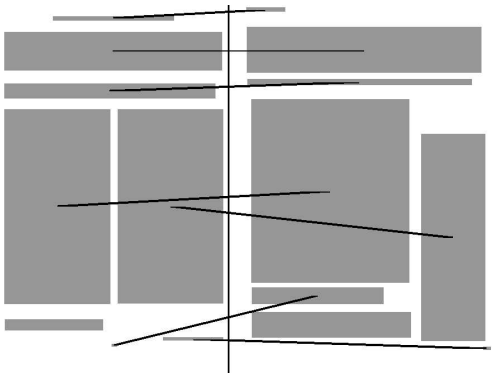
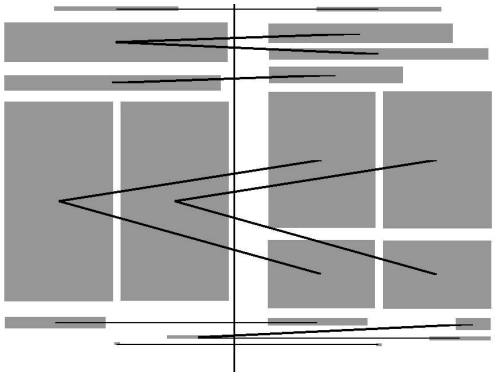


Figure 2.14: Example of a falsely matched journal, although type and publisher are correct. Overlap as block distance and minimum weight edge cover as matching method were used.



(a) Wrong journal, wrong publisher, wrong type



(b) Best correct match

Figure 2.15: Example for a failure. The upper image shows the best match, the lower the best match for a layout of the same journal, type and publisher.

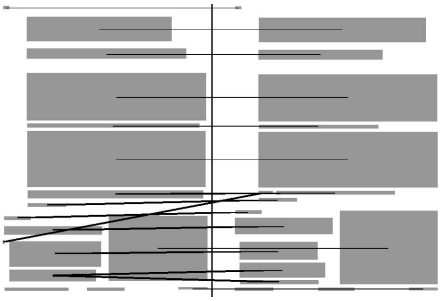
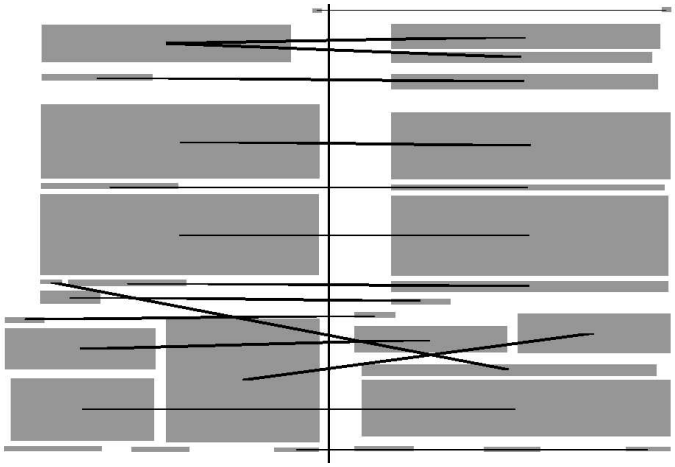
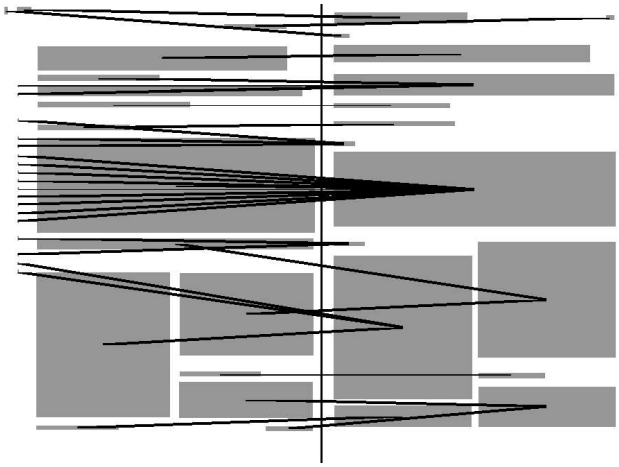


Figure 2.16: Example of an error in the MARG database. According to the database, the type of both documents is different, which is obviously not true.



(a) Journal and type are wrong, publisher is correct



(b) Journal and publisher are wrong, type is correct

Figure 2.17: Examples of errors where either the publisher or the type is wrong.

k-nearest neighbour, where k is greater than 1. So the final match would be a reference layout that, of k similar layouts, has the most layouts from the same journal in this k-neighbourhood. This idea has been tested, results can be found in Table 2.13, but no improvement has been observed for $k = 1, ..., 5$.

2.9 Application: Layout Analysis by Example

In this section we outline a method using the distance measure introduced in the preceding section, that does layout analysis by example. The general idea of layout analysis by example is to segment a new document by using documents for which the layout is already known. One could e.g. take one document of a three column magazine, run a segmentation algorithm on it, take the output and look for a similar layout in the database. Then, the layout of this similar document is used to segment the new document. This could be done by adapting the blocks of the new layout heuristically using the information of the best matching layout, which is not a trivial problem: given two layouts, one new and not error-free layout and one known correct layout. By what methods can one combine the two layouts into one layout that should be more correct as the initial layout given by the segmentation algorithm?

In order to avoid this problem, we propose another more simple approach: we used three page segmentation algorithms, namely Voronoi, Docstrum and Whitespace by Baird to obtain three different segmentations of the same document. These layouts are then compared to a database of known layouts. The layout of the algorithm that has the smallest distance to a document in the database is used as the output of this combined segmentation process. By this method we hope that the different weaknesses of the segmentation algorithms will become less important.

2.9.1 Evaluation

The method has been tested on the "UW 3" database. The 1600 documents of the database has been divided into two sets: 878 test documents and 722 reference documents (the database). The two sets are exactly the same as in [27], in order to be able to compare the performance of this composed method to the single segmentation algorithms Voronoi, Docstrum and Whitespace.

The method to combine the three different segmentation algorithms works as follows:

Err.	¬J ¬T ¬P	¬J ¬T P	¬J T ¬P	¬J T P	Jour	Type	Pub
k = 3	42	24	18	176	31.9%	8.0%	7.4%
k = 4	49	25	19	183	33.9%	9.1%	8.3%
k = 5	58	27	25	191	36.9%	10.4%	10.2%
k = 6	65	23	27	194	76.9%	10.8%	11.2%
k = 7	72	23	29	201	39.9%	11.6%	12.4%

Table 2.13: Number of errors and error rates for the k-nearest neighbours, $k = 3...7$.

- Step 1: Take a document of the test set and segment it using the three page segmentation algorithms: Voronoi, Docstrum and Whitespace by Baird.
- Step 2: Compare the three layouts obtained in step 1 to the ground truth of the reference set using the proposed distance measure. Save the distance of the best match for each of the three layouts (the best match for each of the three layouts may be different!).
- Step 3: Choose the algorithm that produced the layout with the shortest distance to its best match as the winner. The layout produced by this algorithm is the final segmentation.

The evaluation of this method is done by using the results presented in [27]: for each document, the algorithm that gave the best segmentation result was chosen as an optimal solution for the method presented here. Then the error rate of this optimal choice would be 2.8%. Always choosing Voronoi as best algorithm would give an error rate of 5.5%. For Docstrum it would be 6.0% and for Whitespace 9.9%.

2.9.2 Results

The error rates are shown in Table 2.14. The block distance used is the overlapping area combined with the Manhattan distance. As one can see, the method we proposed does not give the improvement hoped for. It is slightly better as simply choosing one of the three layouts at random, but it stays above the error rate one would have if always choosing the Voronoi page segmentation algorithm. A test with the difference in width as block distance gave slightly better results. An error rate of 5.6% was obtained. Other block distances have been tested without any improvement of the error rate. Normalizing the distance by the number of blocks of the new layout has also been tested but did not succed in any improvement. The same is true for using relative coordinates of the blocks instead of absolute ones.

Interpreting these results, different conclusions are possible and further tests have to be done in order to find out what conclusion is the right one:

- Wrong Approach: it may be that the chosen approach is wrong. The approach bases on the hope, that there is some document in the reference set where the ground truth layout is similar to the segmented layout and that then this segmented document is similar to the ground truth. If this is not the case, e.g. if there is a document where the segmented layout is very similar to one layout of the ground truth but the original documents are not the same (this may be due to segmentation errors), then this method will fail.
- Reference Set: it may be that the reference set is to small to contain a good match for the segmented output. As the number of different layouts is even for Manhattan layouts very large, it may happen that there is no layout in the reference set that is similar to the segmented layout.
- Bad Distance Measure: another possible explanation could be that the distance measure for comparing two layouts is not adapted for this type of application. The difference in width giving better results as the overlapping area could be an indication for this hypothesis.

In order to find out if the distance measure works well, the following test has been done: instead of comparing the segmented page to the database, it was only compared to its ground truth. If the distance measure works well, the error rate for this test should be near to the optimal. Considering the error rate of 6.2% for this test, we can conclude, that there is some problem with the distance measure. An example for a match for this test can be found in Figure 2.18. There you can see, that the distance measure has problems handling small blocks that probabely come from noise. Further tests have to be done to find out what exactly causes this approach to fail.

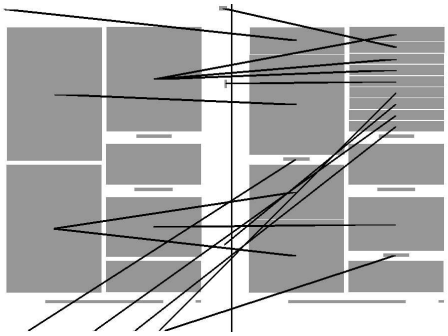


Figure 2.18: Example for comparison of a new document to its ground truth. Small blocks are problematic as their cost is very high to match them, although it would be better to ignore them. N.B.: Missing matching lines are caused by an error in the drawing function (blocks in the middle of the right column are matched but the line is not visible.

Settings	Error Rate
k = 1	6.6
k = 2	6.6
k = 3	6.5
k = 5	6.5
k = 5	6.5
Rel. position, k = 1	6.4
Normalised , k = 1	7.3
Choosing at random	7.0
Block distance: Diff. in width	5.9
Block distance: Diff. in height	6.7
Block distance: Sum of Manh. dist. of corners	6.3
Comparison to ground truth:	6.2
Optimal Choice	2.8
Always Voronoi	5.5
Always Dosctrum	6.0
Always Whitespace	9.9

Table 2.14: The error rates for layout analysis by example in [%].

2.10 Conclusion and Future Work

We presented a distance measure for document layouts. It is a two step method that first computes the distance between blocks and then computes a matching between these blocks. Three different methods were proposed: assignment problem solving, minimum weight edge cover and Earth Mover's Distance. Different block distances have also been proposed and tested together with the matching methods using the publicly available MARG database.

Testing these methods lead us to conclude that the best matching method for document image retrieval by layout is the minimum weight edge cover method, as it gave the lowest error rates. Furthermore we saw that the Earth Mover's Distance is not appropriate for document image retrieval by layout as its time consumption is to high.

For the block distances we observed that the overlapping area is the best choice, as it combines height, width and aspect-ratio into one distance. By combining it with the Manhattan distance of the corner points of the block it could slightly be improved. Concerning the computation time we can conclude, that albeit there are differences for the different block distances, the overall computation time is quite low, as the block distances are all very simple.

Concerning the different error types we saw, that a small number of errors were due to errors in the MARG database. Most of the errors are due to different journals from the same publisher and the same type. Given these facts, it is traceable that the distance measure has problems to keep apart these layouts, as even for a human person, the layouts look similar. The rest of the errors are more problematic errors, as the distance measure in these cases returns layouts that from a human point of view do not look similar. On the bases of which further investigation can be done to find out how to improve the algorithm.

An application for the distance measure, namely layout analysis by example was presented, too. Although the results were not as positive as hoped, different hypothesis for these high error rates could be established and further examination is needed in order to find out if the approach is a possible step to improve page segmentation algorithms. If not, other approaches are possible: using the method described by Breuel in [6] to build a hierarchical tree of all possible layouts, the distance measure could be used to find the best match of all these possible segmentations and use that one as segmentation to choose.

Chapter 3

Layout to HTML Converter

Extracting geometric layout information is an important process step in layout analysis. In order to visualise the results of such an algorithm we are interested in a tool that allows us to check these in a simple manner.

Furthermore it would be interesting to have a common interface for layout information that allows, apart from simple visualisation, also further processing as well. HTML tables have been chosen as a good format that allows to visualise the results, to present them on the net and that is a wide-spread language.

This chapter is divided into three parts: the first part contains a short description about the requirements for this program. The second part explains the different functionalities, and the last part gives some information about the implementation details.

3.1 Requirements

In this part, the different exigencies to the tool are described. The tool should be able to do the following things:

- Manual Segmentation: The tool should assist the user in manually segmenting a document image into different zones. This should be done by the XY-Cut manner of splitting recursively horizontally and vertically to achieve the desired segmentation. The user should have the possibility to cut at a certain position in the image, simply by clicking on it. Left mouse clicks should produce horizontal cuts, right mouse clicks vertical cuts. The results of each cut should be displayed immediately, so that the user can verify his segmentation.

- Export to HTML: To display the segmentation in a widely used format, the tool should be able to convert the segmentation to HTML code. The XY-Cut method of splitting the page recursively into various parts perfectly fits into the concept of nested HTML tables that can be built recursively, too. The images, that are the result of the split original image, should be placed in the right table to guarantee that the resulting HTML code produces the same layout as the original image. Furthermore, white borders in the partial images should be removed to reduce the space needed to save partial images. The tool should also allow drawing the HTML borders to

visualise the structure of the HTML table and the segmentation. For PDF files it should paste the text into the concerning table cells instead of the image.

- Import of segmentation information: As a wide variety of layout analysis algorithms exists and their code is also available, the tool should offer an interface to import the output of these algorithms. As most algorithms output blocks, and not zones as does XY-Cut, the tool should be able to import this block data and display it on the original image. It also should be able to convert this block data into an HTML table, although the conversion from blocks to tables is not trivial.

- Usability: The tool should be easy to use and provide a few little useful functionalities for the user, as e.g. undoing, redoing, zooming in and zooming out and displaying crosshairs to simplify the positioning of the cut to be done.

- Automatic Segmentation: As manual segmentation can be time consuming, the tool should also provide a simple method for automatic segmentation. This segmentation should use XY-Cut algorithm.

- Input file format: As there is a large number of different file formats for images, a few of them had to be chosen. TIF and PPM should be readable by the tool, as TIF and PPM are a very wide spread format for document images. In addition, PDF files should be a possible input. For these, the original text should be extracted to paste it onto the HTML table.

3.2 Functionalities and Description of the Graphical User Interface

3.2.1 Mouse Buttons: Manual Segmentation

The manual segmentation consists of cutting the page into different parts by splitting it horizontally and vertically at different positions, just like the XY-Cut algorithm does. An example for consecutive horizontal and vertical cuts can be found in Figure 3.1.

The splitting is done by selecting a position in the image and then clicking the left mouse button for a horizontal split and the right mouse button for a vertical split. After each split the new segmentation is drawn in green lines. To facilitate the positioning of the splitting, crosshairs can be activated. A screenshot of the segmentation process can be seen in Figure 3.2.

3.2.2 Menu entries

There are only two menus, the “File” and the “Edit” menu. The “File” menu contains the usual items to open and close a file, to quit the program and to export the segmented page to HTML. The “Edit” menu offers a method for undoing and redoing segmentation steps and a possibility to activate crosshairs. In the following, the menu entries are explained in more detail.

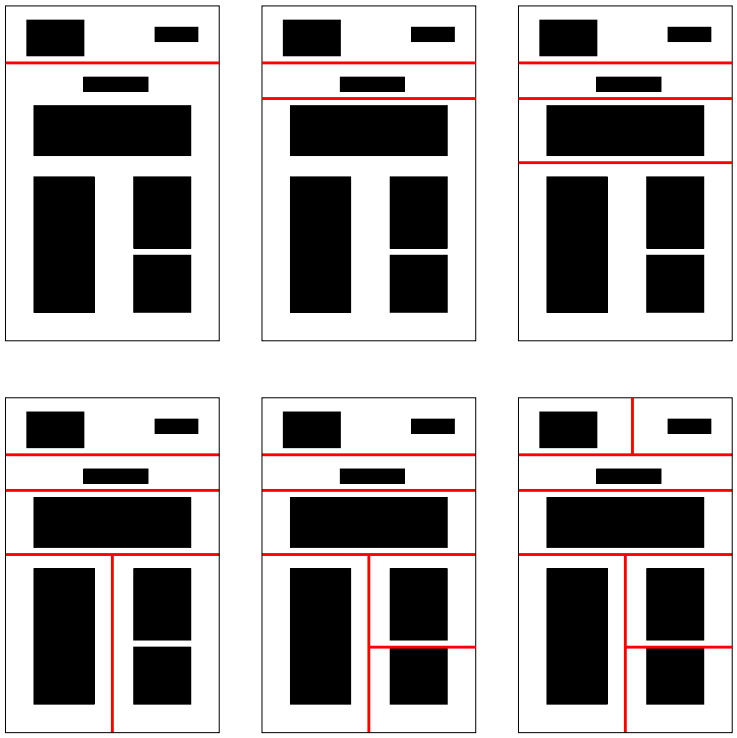


Figure 3.1: An example for consecutive cuts that segment a page into different zones.

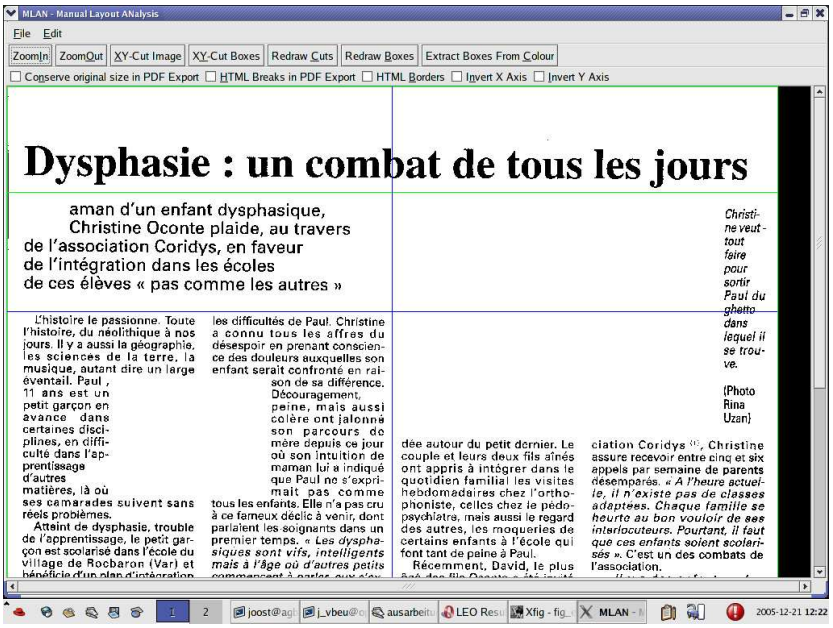


Figure 3.2: A screenshot of the manual segmentation process. The green line represents a horizontal cut, the blue lines represent the crosshairs.

Open

The open dialog lets you choose the file the user wants to open. Supported file formats are:

- TIF, PPM: input image to be segmented and converted to HTML
- PDF: input PDF file to be segmented and converted to HTML
- DAT: text file containing coordinates of the corner points in $x_l\ y_l\ x_h\ y_h$ integer format of the blocks, where (x_l, y_l) defines the lower left and (x_h, y_h) the upper right corner of the block. Choosing a DAT file opens a new "Open" dialog where the user has to choose the image this block information belongs to. There is no check whether it is the right file or not, nor a check if the DAT file has the right format; the user has to take care of it. An example for external block data displayed on the original document can be found in Figure 3.3

Close, Quit

The "Close" function resets the graphical user interface to the initial state and internally resets all the data structures so that every segmentation information is lost. "Quit" closes the program.

Export as HTML

The "Export as HTML" function fulfils the main requirement of the program: it converts a set of horizontal and vertical cuts into an HTML table. This

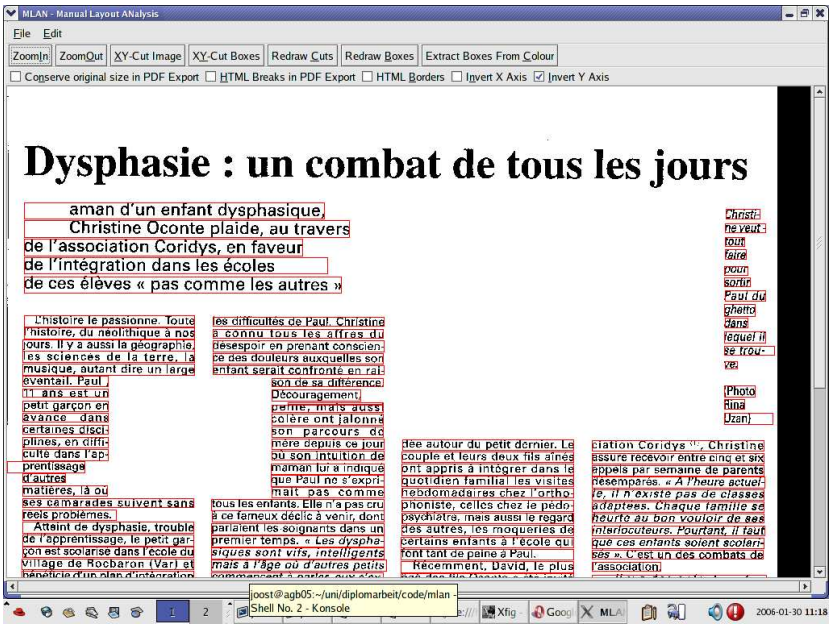


Figure 3.3: A screenshot of the GUI displaying block information (red blocks) on the original document image.

conversion process is described in the following.

Starting with the current segmentation given by the cuts, done by the user or obtained by the automatic segmentation, the export method builds a nested HTML table that looks the same as the segmented image. A nested HTML table is a table that contains one or more other tables, that in turn also can contain tables, and so on. The original image is split into different subimages and these are then pasted into the different table cells.

An example of an HTML version of a segmented image can be found in Figure 3.4. In Figure 3.5 the same segmentation is shown, this time with HTML table borders.

Undo, Redo, Crosshair

"Undo" removes the currently last cut from the list and redraws the blocks we obtain. "Redo" moves the recently removed cut back to the list and redraws the blocks. "Crosshairs" activates or deactivates the crosshair.

Figure 3.4: A screenshot of an HTML version of a segmented page in a browser window.

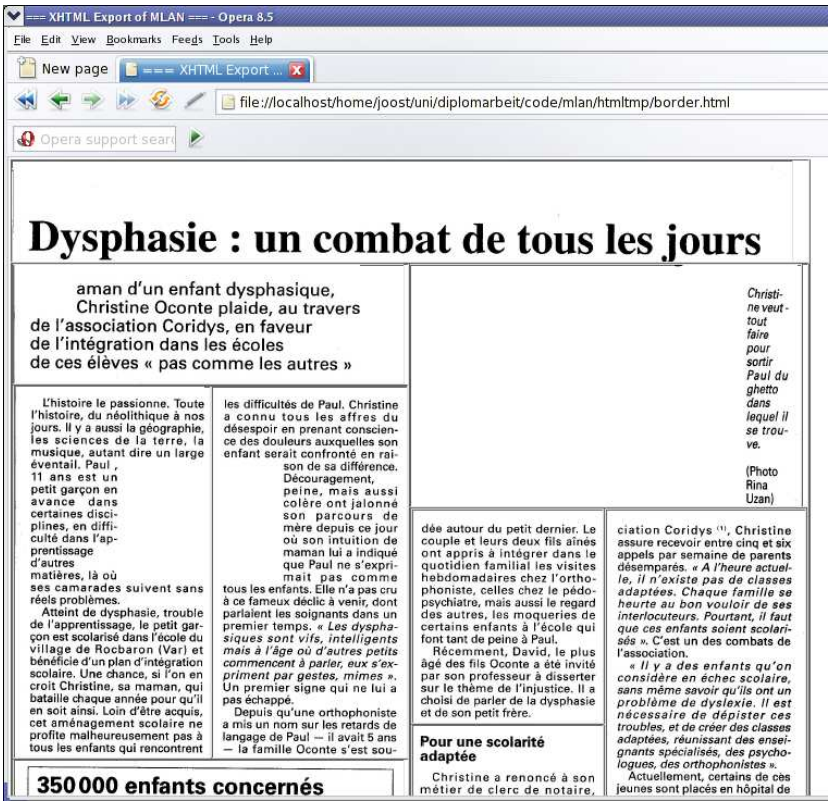


Figure 3.5: A screenshot of an HTML version with borders of a segmented page in a browser window.

3.2.3 Buttons

A screenshot of the buttons can be found in Figure 3.6.

Zoom in, Zoom out

These buttons are used to zoom in and to zoom out. The factor of zooming is diminished or increased by 0.1 per step. The minimal factor is 0.2 and the maximal 10.0.

XY-Cut Image

Clicking this button starts the XY-Cut segmentation algorithm. This algorithm recursively splits the page in horizontal or vertical direction. The coordinates where it should split are found by finding the largest possible gaps in the X and Y projection of the image. A more detailed description of this algorithm can be found in Section 2.7.2. The result of the segmentation method is then displayed on the screen. The user has the possibility to manipulate the obtained segmentation by undoing segmentation steps and by adding new cuts.

XY-Cut Boxes

This method is used to convert the output blocks of some general layout segmentation algorithms to a HTML table. Because converting some arbitrary blocks to a table is not that trivial, the following method was chosen for this task:

First, an image is created in which the blocks are simply painted black. Then, the XY-Cut algorithm splits the image into different regions that can be converted without problem to a nested HTML table. An illustration of this procedure can be found in Figure 3.7.

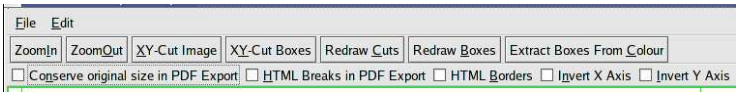


Figure 3.6: A screenshot of the menubar of the HTML conversion tool.

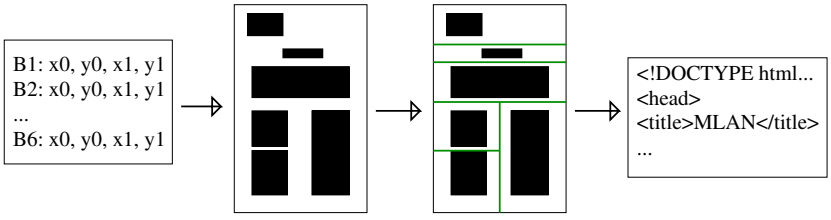


Figure 3.7: An illustration of the conversion from block data to an HTML table. On the left side we have the coordinates of the blocks obtained by some segmentation algorithm. These are converted to an image containing black blocks. On this image we then run the XY-Cut algorithm. The last step then converts the result of the XY-Cut algorithm to HTML code.

Redraw Blocks

When blocks from another segmentation layout are used as input, these are also represented on the image. If the user now wants to segment the page, the program automatically changes the presentation and draws the regions obtained by the cuts. By pressing this button, the user will see the original blocks from the input data.

Redraw Cuts

This button has the inverted functionality of the preceding one: instead of changing the presentation from cuts to blocks, this button changes the representation from blocks to cuts. This can be used if the user wants to check the result of the automatic segmentation with the input blocks.

Extract Boxes from Colour

Instead of saving the output blocks from layout segmentation process in separate (text) files, one could also save this information by coding the different blocks in different colours. So each block will be identified by its own specific colour. This function finds the minimum bounding box for all the pixels of the different colour values. So if a bitone image is used, it returns the minimum bounding box of all the black pixels in the page.

3.2.4 Checkboxes

This section describes the functionality of the checkboxes in the tool bar.

Conserve Original Size in HTML Export

This checkbox is only useful when converting a PDF document. If this checkbox is enabled, the program tries to conserve the original size of the tables in HTML export. In the other case, the program does not specify the size of the text cells but simply uses the normal text flow to define the size of the blocks.

HTML Breaks in HTML Export

This checkbox is only useful when converting a PDF document. Enabling this checkbox will bring the program to insert
 at the end of the line. In the other case it will simply let the text flow decide when to insert a new line.

HTML Borders

This option sets the border size of the HTML tables to some value greater than zero. In the normal case, the borders have size 0 in order to maintain the original aspect of the document if the HTML conversion is looked in a browser.

Invert X Axis, Invert Y Axis

These two checkboxes allow to invert the axis of the coordinate system. This is needed as the coordinate system of images may differ. If the user then loads a

DAT file with blocks, the blocks will be in the wrong place. Inverting the axis will solve this problem.

3.3 Handling PDF Files

As implementing a parser for PDF files is very time consuming and not trivial and as no free PDF parser library is available, we chose the "pdftotext" program to extract the text. A little modification had to be done in order to output the coordinates of the characters and not only the characters.

When the user opens a PDF file, he will be asked to chose the page number he wants to convert. Then this page will be converted to PNM in background and the text together with the coordinate information is extracted. Now the user can segment the page. When exporting it to HTML, the text, instead of the image, is put into the table. If one cell of the table contains no character, we presume that there should be an image and copy the corresponding part of the image into that cell. As we are removing the white borders of the partial images before saving, entirely white images will be recognised and not saved.

An example of the HTML version of a segmented PDF page can be found in Figure 3.8.

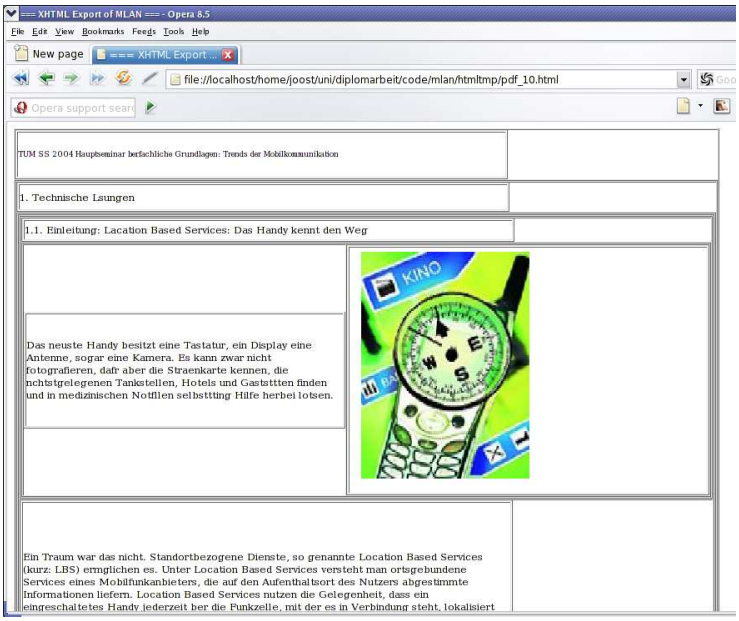


Figure 3.8: A screenshot of the HTML version of a segmented PDF page. The border option is turned on and no breaks are inserted.

Appendix A

Matching Algorithms

A.1 Hungarian Algorithm

A detailed description of the algorithm can be found in [17]. The algorithm for solving the assignment problem works as follows (the description and the example are taken from [24]):

- Step 0: If necessary, rotate the cost matrix so that the number of rows is less or equal to the number of columns; $k = min(n, m)$, where n equals the number of rows and m the number of columns of the assignment problem table.
- Step 1: $\forall$ row: find the smallest element and subtract it from every element in this row.
- Step 2: Find a cell containing a zero, called Z: if in the row or column of Z there is no starred zero, Z will be starred. Repeat this step for all elements in the matrix.
- Step 3: Cover all columns containing a starred zero. If the number of covered columns equals k, $k = min(n, m)$, then we are finished, Goto Done. Else continue with step 4.
- Step 4: Find an uncovered zero and prime it (Z'). If no starred zero Z* can be found in the row of Z', Goto step 5. Else cover the row of Z' and uncover the column of Z*. Continue until no uncovered zeros are left. Save smallest uncovered value and goto step 6.
- Step 5: Starting at the Z' found in step 4, we build a path by alternatively looking for a starred zero in the column and for a primed zero in the row. Continue until we find a primed zero where no starred zero can be found in the same column. For each zero of the path, change starred zeros to unstarred zeros and change primed zeros to starred zeros. Delete every prime and uncover every line. Goto step 3
- Step 6: Add the value of step 4 to every element of every uncovered row. Subtract the value from every element of every uncovered column. Goto step 4

- Done: A starred zero in cell i, j indicates that the block in row i is assigned to block in column j.

An example for the different steps of this algorithm can be found in Table A.1.

A.2 Minimum Weight Edge Cover Algorithm

The steps to solve the minimum weight edge cover problem using the Hungarian algorithm, are the following (the description of the algorithm is from [15] and from [14]):

- Step 1: find for each node the edge with the minimum weight
- Step 2: for each possible edge, subtract the minimum weights of the two nodes of this edge
- Step 3: find the minimum value of the matrix obtained in step 2 and subtract this from every value in order to obtain only positive values. Apply the Hungarian algorithm.
- Step 4: from the result of the Hungarian algorithm, delete every edge that has a nonzero weight
- Step 5: for each unmatched node add an edge with minimum weight to the cover

An example for this method can be found in Table A.2.

A.3 Transportation Problem Solving Algorithm

As mentioned before, the transportation problem has to be solved in order to compute the EMD. A mathematically more profound approach can be found in [3]. To explain the algorithm the balanced case is considered. This means that the sum of all the demands equals the sum of all the supplies. Although this condition is not always satisfied for signatures, this does not present a problem, as in this case either a dummy supply or a dummy demand row can be added. This dummy produces or consumes the difference of supplies and demands between the two signatures. If demand and supply are equal, the problem is called "balanced", the other case "unbalanced" transportation problem.

Solving the transportation problem is done in two major steps: the first step consists of finding an initial feasible solution, The second step is optimising this solution. For the first part the minimum cost method is used to find a feasible solution. For the second part the transportation simplex method is used.

A.3.1 Minimum Cost Method for Finding a Feasible Solution

A sketch of the algorithm is given here:

	B1	B2	B3
Ba	1	2	3
Bb	2	4	6
Bc	3	6	9

Initial Cost Matrix

	B1	B2	B3
Ba	0	1	2
Bb	0	2	3
Bc	0	3	6

After Step 1

	B1	B2	B3
Ba	0*	1	2
Bb	0	2	3
Bc	0	3	6

After Step 2

	B1	B2	B3
Ba	*0**	1	2
Bb	*0*	2	3
Bc	*0*	3	6

After Step 3

	B1	B2	B3
Ba	*0**	**1**	2
Bb	*0*	2	3
Bc	*0*	3	6

After Step 4

	B1	B2	B3
Ba	*0**	0	1
Bb	*0*	1	3
Bc	*0*	2	5

After Step 6

	B1	B2	B3
Ba	*0**	*0´*	*1*
Bb	0´	1	3
Bc	0	2	5

After Step 4

	B1	B2	B3
Ba	***0****	***0´***	*1*
Bb	**0´**	1	3
Bc	0	2	5

After Step 5

	B1	*B2*	B3
Ba	*0*	*0**	1
Bb	*0**	*1*	3
Bc	*0*	*2*	5

After Step 3

	B1	*B2*	B3
Ba	*0*	*0**	**1**
Bb	*0**	*1*	3
Bc	*0*	*2*	5

After Step 4

	B1	*B2*	B3
Ba	*0*	*0**	0
Bb	*0**	*1*	2
Bc	*0*	*2*	4

After Step 6

	B1	B2	B3
Ba	*0*	*0**	*0´*
Bb	*0**	**1**	2
Bc	*0*	2	4

After Step 4

	B1	B2	B3
Ba	*1*	*0**	*0´*
Bb	*0**	0	1
Bc	*0*	1	3

After Step 6

	B1	B2	B3
Ba	*1*	*0**	*0´*
Bb	*0**	*0´*	*1*
Bc	*0´*	1	3

After Step 4

	B1	B2	B3
Ba	1	**0***	**0´**
Bb	**0***	**0´**	1
Bc	**0´**	1	3

After Step 5

	B1	*B2*	*B3*
Ba	*1*	*0*	*0**
Bb	*0*	*0**	*1*
Bc	*0**	*1*	*3*

After Step 3

	B1	B2	B3
Ba	1	0	0*
Bb	0	0*	1
Bc	0*	1	3

Done

Table A.1: An example for the different steps of the Hungarian algorithm for solving the assignment problem. Rows or columns in *italic* symbolise the covering of rows or columns. Values in **bold** are marked cells that are needed in further steps.

	B1	B2	B3	B4
Ba	1	2	3	4
Bb	2	4	6	8
Bc	3	6	9	12

Initial Cost Matrix

	B1	B2	B3	B4
Ba	**1**	**2**	**3**	**4**
Bb	**2**	4	6	8
Bc	**3**	6	9	12

Minima found in step 1

	B1	B2	B3	B4
Ba	-1	-1	-1	-1
Bb	-1	0	1	2
Bc	-1	1	3	5

Step 2: after subtracting the minima

	B1	B2	B3	B4
Ba	0	0	0	0
Bb	0	1	2	3
Bc	0	2	4	5

After step 3

	B1	B2	B3	B4
Ba	0	0	1	0
Bb	0	1	0	0
Bc	1	0	0	0

Result of Hungarian Algorithm

	B1	B2	B3	B4
Ba	0	0	1	1
Bb	0	1	0	0
Bc	1	0	0	0

After step 5

Table A.2: An example for minimum weight edge cover problem algorithm based on the Hungarian algorithm.

- Step 1: Get the cell with the least costs and allocate the most possible resources, given by the minimum of demand and supply for this cell. Reduce demand and supply in this column and row by the allocated amount. If total demand is greater than zero, repeat step 1. Else stop.

This method starts with looking for the cell with the least costs to allocate one unit (this means that one unit of goods from a specified producer is assigned to a consumer). After having found this cell, the maximum of units that can be allocated is determined. This value is given by the minimum of the supply of this row and the demand of this column. Then the supply and demand for this row and this column are reduced by this minimum and allocated to the cell. We now have allocated a transport of $min(supp(i), dem(j))$ units from $F(i)$ to $C(j)$, and the remaining demands and supplies were adjusted accordingly. This process is repeated until no demand and no supply remains (keeping in mind that only balanced problems are considered, both demand and supply come simultaneously to zero).

Other methods to find this initial feasible solution are possible. As the number of iterations of the next part of the algorithm depends on this first solution, it is of interest to find a good feasible solution (a solution that is as close to the optimum as possible). This is the reason why we chose the minimum cost method. Other possibilities to find an initial feasible solution are the Northwest corner method and Vogel's approximation model. Descriptions of these methods can be found in [1]. An example can be found in Table A.8. Variants of Vogelś method can be found in [20].

A.3.2 Optimisation of the Initial Solution

After having found an initial feasible solution it has to be optimised in order to get an optimal solution. From the first step we get an array that contains allocation values in different cells. An example can be found in Table A.3. The cells having an allocation that is greater than zero will be called "basic cells", the other cells will be called "non-basic cells" (other terms used are "non-empty" and "empty" cells or "occupied" and "non-occupied"cells).

The algorithm now optimises this first solution by repeating the following three steps:

- Step 1: For each non-basic cell, compute the reduced costs. Select the non-basic cell with the most negative reduced cost, in other words, the cell with the lowest reduced cost smaller then zero. If no such cell exists, then the optimal solution is found.
- Step 2: Generate a stepping stone path from the cell we found in step 1 to the same cell by walking over basic cells.

	C1	C2	C3	Dummy	Remaining Supp.
F1	**25** *24*	**5** *30*	**0** *40*	**20** *0*	0
F2	**0** *30*	**40** *40*	**10** *42*	**0** *0*	0
Remaining Dem.	0	0	0	0	

Table A.3: The solution found using minimum cost method. Values in *italic* denote the costs for this cell, values in **bold** the allocation made for this cell.

- Step 3: Determine the minimum possible allocation for this path. Update the allocation matrix by adding or subtracting the minimum possible allocation to the cells on the path. The basic cell with the minimum possible allocation is changed to a non-basic cell as its allocation becomes zero. If more than one basic cell on the path is degraded to zero, choose arbitrarily one that will be changed to a non-basic cell. The non-basic cell we started with becomes a basic cell. Continue with step 1.

Step 1: Computing reduced costs

This step uses as input the solution obtained from the minimum cost method or from the preceding iteration and calculates the reduced costs matrix. This matrix gives an overview how the total costs change if the allocations are changed (e.g. units from one producer are assigned to a different consumer). The reduced cost matrix, that is obtained in this step, indicates for each non-basic cell how much costs can be saved by assigning one unit of goods to this non-basic cell. This method works as follows:

Let u_i be a variable associated with row i and v_j a variable associated with column j of the transportation problem matrix (we only consider the "cost" cells, not the last column nor the last row that contain only the supplier and consumer information). The number of variables is $n+m$, where n is the number of blocks of layout L_1 and m is the number of blocks of layout L_2.

The relationship for basic cells is given by: $c_{i,j} = u_i + v_j$, where $c_{i,j}$ is the cost of the cell in row i and column j. The relationship for non-basic cells is given by: $rc_{i,j} = c_{i,j} - u_i - v_j$, where $rc_{i,j}$ is the reduced cost of the non-basic cell in row i and column j.

We then obtain a set of linear equations with $n+m-1$ equations and $n+m$ of variables. In order to be able to solve this system, the value of u_i is set to zero: $u_i = 0$. Now the linear set of equations can be solved and a value for each u_i and v_j is gained and the reduced costs matrix is computed. An example for the reduced costs matrix can be found in Table A.4.

Step 2: Stepping Stone Path Finding Method

After having computed the reduced cost matrix the non-basic cell with the most negative reduced cost is chosen. This value indicates that the total costs of the solution will be reduced by this value for every unit allocated to this cell. So we are interested to choose the cell with the most negative reduced cost. And if

	C1	C2	C3	Dummy	u_i
F1	*24*	*30*	**+8** *40*	*0*	0
F2	**-4** *30*	*40*	*42*	**-10** *0*	10
v_i	24	30	32	0	

Table A.4: The reduced cost matrix of the initial solution. Values in *italic* denote the costs for this cell, values in **bold** the reduced costs for the non-basic cell. Assigning one unit of goods to cell (F2, Dummy) would decrease the total cost by 10.

there is no negative reduced cost cell, the solution we have cannot be improved and we have an optimal solution.

In order to allocate one unit to this cell, units from other basic cell have to be removed, as it is not allowed to change the total demand or supply of one row or column, as these are given as part of the problem. Removing one unit in this cell implies adding one unit in another cell in order to meet the constraints of total demand and and total supply for each row and column. So a stepping stone path is needed that starts and ends in the cell with the most negative number and that only passes basic cells (except for the starting cell). An example can be found in Figure A.1. An example of a stepping stone path for the example above can be found in Table A.5.

The algorithm works as follows:

- Step 1: Mark the starting cell with a "+" sign
- Step 2: Check for a basic unvisited cell in the same row or in the same column. If there is one, go to step 3. Else go to step 4.
- Step 3: If the cell is the starting cell and the path has a length greater than 3, stop. Else mark the cell as visited, add it to the path and mark it with the correct sign: "+", if the sign of the last cell was a "-" and "-" if the sign of the last cell on the path was as "+".
- Step 4: Go one step back on the path. Go to step 2.

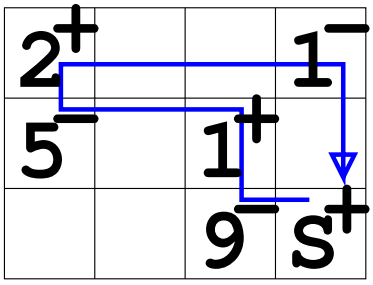


Figure A.1: Example of a stepping stone path. The "S" cell is the starting point. The blue line gives the stepping stone path. The Numbers in the cell are the assignments of the basic cells. The maximum possible new assignment is 1.

	C1	C2	C3	Dummy	u_i
F1	**25**	**5** *"+"*	**0**	**20** *"-"*	0
F2	**0**	**40** *"-"*	**10** *0*	**0** *Start* *"+"*	10
v_i	24	30	32	0	

Table A.5: The reduced cost matrix of the initial solution. Values in *italic* denote the sign for the cell (if it is part of the path, in the other case it is set to 0), values in **bold** are the current allocations of the cell.

Step 3: Updating the Assignment Matrix

After having found the path, we look for the minimum of all the allocations in basic cells made on this path. This minimum is then added or subtracted to all the cells that are part of the path, depending on the sign the cell obtained during the stepping stone path finding method. So the starting cell will be transformed from a non-basic to a basic cell and one cell on the path will be transformed from a basic to a non-basic cell because its allocation will become zero. If the allocation of more than one basic cell is reduced to zero only one of these cells is changed to a non-basic cell. Now we continue with Step 1.

When the optimisation of the solution is finished, that means when the reduced costs matrix obtained after step 1 has no more negative number in any of its cells, the total costs of the optimal solution can be computed by summing up the product of the allocation of every cell with its cost.

A.3.3 An Example for Solving the Transportation Problem

Given the following transportation problem

Example, Part 1: Initial Solution:

Given the transportation tableau in Table A.6 After the first iteration of the minimum cost method we obtain the following tableau:

Example, Part 2: Finding the optimal solution

After obtaining a first feasible solution we optimise the solution progressively:

The total cost C_t is obtained by summing up the product of the allocation times the costs for each cell: $C_t = 5{\times}24{+}45{\times}40{+}20{\times}30{+}10{\times}42{+}20{\times}0 = 2490$.

	C1	C2	C3	Dummy	Supply
F1	24	30	40	0	50
F2	30	40	42	0	50
Demand	25	45	10	20	

Table A.6: The transportation tableau

	C1	C2	C3	Dummy	Supply
F1	**0** *24*	**0** *30*	**0** *40*	**20** *0*	30
F2	**0** *30*	**0** *40*	**0** *42*	**0** *0*	50
Demand	25	45	10	0	

After second iteration:

	C1	C2	C3	Dummy	Supply
F1	**25** *24*	**0** *30*	**0** *40*	**20** *0*	5
F2	**0** *30*	**0** *40*	**0** *42*	**0** *0*	50
Demand	0	45	10	0	

After third iteration:

	C1	C2	C3	Dummy	Supply
F1	**25** *24*	**5** *30*	**0** *40*	**20** *0*	0
F2	**0** *30*	**0** *40*	**0** *42*	**0** *0*	50
Demand	0	40	10	0	

After fourth iteration:

	C1	C2	C3	Dummy	Supply
F1	**25** *24*	**5** *30*	**0** *40*	**20** *0*	0
F2	**0** *30*	**40** *40*	**0** *42*	**0** *0*	10
Demand	0	0	10	0	

After fifth and last iteration:

	C1	C2	C3	Dummy	Supply
F1	**25** *24*	**5** *30*	**0** *40*	**20** *0*	0
F2	**0** *30*	**40** *40*	**10** *42*	**0** *0*	0
Demand	0	0	0	0	

Table A.7: The minimum cost method. Values in *italic* denote the costs for this cell, values in **bold** the allocation made for this cell.

1. Iteration:

Computing reduced costs:

	C1	C2	C3	Dummy	u_i
F1	**25** *0*	**5** *0*	**0** *+8*	**20** *0*	0
F2	**0** *-4*	**40** *0*	**10** *0*	**0** *-10*	10
v_i	24	30	32	0	

Select cell (F2, Dummy) as starting cell for the stepping stone path.

Stepping Stone path:
c(2,4) "+" → c(1,4) "-" → c(1,2) "+" → c(2,2) "-"
Minimum possible allocation on this path: 20 units.

Then we update the allocations and compute the new reduced costs:

2. Iteration:

	C1	C2	C3	Dummy	u_i
F1	**25** *0*	**25** *0*	**0** *+8*	**0** *+10*	0
F2	**0** *-4*	**20** *0*	**10** *0*	**20** *0*	10
v_i	24	30	36	-6	

Select cell (F2, C1) as starting cell for the stepping stone path.

Stepping Stone path:
c(2,1) "+" → c(1,1) "-" → c(1,2) "+" → c(2,2) "-"
Minimum possible allocation on this path: 20 units.

Then we update the allocations and compute the new reduced costs:

3. Iteration:

	C1	C2	C3	Dummy	u_i
F1	**5** *0*	**45** *0*	**0** *+4*	**0** *+6*	0
F2	**20** *0*	**0** *+4*	**10** *0*	**20** *0*	10
v_i	24	30	36	-6	

STOP, as no non-basic cell has a reduced cost less than zero.

Table A.8: The transportation simplex method. Values in *italic* denote the reduced costs for this cell, values in **bold** the allocation made for this cell.

List of Figures

List of Tables

Bibliography

[1] N. F. Angel. Transportation and Assignment Solution Methods. http://www.ferrum.edu/fangel/qm/module_b.pdf, January 2006.

[2] H. S. Baird. Background Structure in Document Images. *Bunke, H. and Wang, P. S. P. and Baird, H. S. (Eds.), Document Image Analysis, World Scientific, Singapore*, pages 17–34, 1994.

[3] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. Wiley, 1990.

[4] T. M. Breuel. Two Geometric Algorithms for Layout Analysis. In *DAS '02: Proceedings of the 5th International Workshop on Document Analysis Systems V*, pages 188–199, London, UK, 2002. Springer-Verlag.

[5] T. M. Breuel. High Performance Document Layout Analysis. In *Symposium on Document Image Understanding Technology, Greenbelt, Maryland*, 2003.

[6] T.M. Breuel. Layout Analysis by Exploring the Space of Segmentation Parameters. In *Proceedings of the International Association for Pattern Recognition Workshop (Document Analysis Systems)*, 2000.

[7] R. Cattoni, T. Coianiz, S. Messelodi, and C. M. Modena. Geometric Layout Analysis Techniques for Document Image Understanding: a Review. Technical report, IRST, Trento, Italy, 1998.

[8] T. Deselaers, D. Keysers, and H. Ney. Features for Image Retrieval – A Quantitative Comparison. In *DAGM 2004, Pattern Recognition, 26th DAGM Symposium*, LNCS, Tbingen, Germany, September 2004.

[9] D. Doermann. The Indexing and Retrieval of Document Images: A Survey. Technical Report LAMP-TR-013,CFAR-TR-878,CS-TR-3876, University of Maryland, College Park, February 1998.

[10] V. Eglin and S. Bres. Document Page Similarity Based on Layout Visual Saliency: Application to Query by Example and Document Classification. In *Seventh International Conference on Document Analysis and Recognition (ICDAR'03)*, volume 2, pages 1208–1212, August 2003.

[11] G. Ford and G. R. Thoma. Ground Truth Data for Document Image Analysis. In *Proceedings of the 2003 Symposium on Document Image Understanding and Technology*, pages 199–205, April 2003.

[12] J. Hu, R. Kashi, and G. T. Wilfong. Document Classification Using Layout Analysis. In *DEXA Workshop*, pages 556–560, September 1999.

[13] J. Hu, R. Kashi, and G. T. Wilfong. Document Image Layout Comparison and Classification. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'99)*, pages 285–288, September 1999.

[14] J. Keijsper and R. Pendavingh. An Efficient Algorithm for Minimum-Weight Bibranching. *Journal of Combinatorial Theory. Series B. Graph Theory and Matroid Theory*, 2:130–145, 1998.

[15] D. Keysers, T. Deselaers, and H. Ney. Pixel-to-Pixel Matching for Image Recognition using Hungarian Graph Matching. In *DAGM 2004, Pattern Recognition, 26th DAGM Symposium*, volume 3175 of *Lecture Notes in Computer Science*, pages 154–162, Tübingen, Germany, August 2004.

[16] K. Kise, A. Sato, and M. Iwata. Segmentation of Page Images Using the Area Voronoi Diagram. *Comput. Vis. Image Underst.*, 70(3):370–382, 1998.

[17] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. Addison-Wesley, Reading, MA, 1994.

[18] J.L. Liang, I.T. Phillips, and R.M. Haralick. Performance Evaluation of Document Structure Extraction Algorithms. *Computer Vision and Image Understanding*, 84(1):144–159, October 2001.

[19] S. Mao and T. Kanungo. Empirical Performance Evaluation Methodology and Its Application to Page Segmentation Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):242–256, 2001.

[20] M. Mathirajan and B. Meenakshi. Experimental Analysis of some Variants of Vogel's Approximation Method. *Asia-Pacific Journal of Operational Research*, 21(4), 2004.

[21] G. Nagy, S. Seth, and M. Viswanathan. A Prototype Document Image Analysis System for Technical Journals. *Computer*, 7(25):10–22, 1992.

[22] H. Q. Ngo. Lecture 1: Matchings on Bipartite Graphs. http://www.cse.buffalo.edu/~hungngo/teaching.html, 2004.

[23] L. O'Gorman. The Document Spectrum for Page Layout Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1162–1173, 1993.

[24] B. Pilgrim. CSC 445 Computer Algorithms. http://216.249.163.93/bob.pilgrim/445/munkres.html, 2005.

[25] Y. Rubner, L. Guibas, and C. Tomassi. The Earth Mover's Distance, Multi-Dimensional Scaling, and Color-Based Image Retrieval. *Proceedings of the DARPA Image Understanding Workshop*, pages 661–668, May 1997.

[26] Y. Rubner, C. Tomasi, and L. J. Guibas. The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[27] F. Shafait, D. Keysers, and T. M. Breuel. Performance Comparison of Six Algorithms for Page Segmentation. In *7th IAPR Workshop on Document Analysis Systems (DAS)*, volume 3872 of *LNCS*, pages 368–379, Nelson, New Zealand, Feb 2006. Springer.

[28] C. Shin, D. Doermann, and A. Rosenfeld. Classification of Document Pages Using Structure-Based Features. *International Journal on Document Analysis and Recognition*, 3(4):232–247, 2001.

[29] H. Tamura, S. Mori, and T. Yamawaki. Textual Features Corresponding to Visual Perception. In *IEEE Transactions on Systems, Man and Cybernetics*, pages 406–472, June 1978.

[30] K. Y. Wong, R. G. Casey, and F. M. Wahl. Document Analysis System. *IBM Journal of Research and Development*, 26(6):647–656, November 1982.

[31] B. A. Yanikoglu and L. Vincent. Pink Panther: A Complete Environment for Ground-Truthing and Benchmarking Document Page Segmentation. *Pattern Recognition*, 31(9):1191–1204, September 1998.