TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

MASTERARBEIT

# Evaluation and Reconstruction of Strip-Shredded Text Documents

ausgeführt am

Institut für Computergrafik und Algorithmen

der Technischen Universität Wien

unter der Anleitung von

Univ.Prof. Dipl.-Ing. Dr. Günther Raidl

und

Univ.Ass. Mag. Dipl.-Ing. Matthias Prandtstetter

durch

Wolfgang Morandell Bakk.techn.

Weinberggasse 53/16

A-1190, Wien

Mai, 2008

## Abstract

In my master's thesis I elaborate on strip-shredded text document reconstruction. Contrary to conventional document reconstruction – which uses color or shape information of images – text document reconstruction has not been researched very well. Nowadays it is common to destroy paper documents by shredding them, i.e. producing paper strips. This work tries to find ways to undo the process. First and foremost I describe the problem formally. Next I define a way to evaluate problem instances. A set of improvement strategies are introduced which help the evaluation process. Defined construction heuristics yield good results in reasonable amount of time. Then optimization algorithms try to find a good arrangement of the strips, ideally the correct one. A demo application simulates the shredding process of a sample page. Then this page is reconstructed using the above mentioned evaluation techniques and several optimization techniques like multistart variable neighborhood search, simulated annealing and iterated local search. Extensive tests were run with a 60 instance test set. The implemented application reconstructed more than half of the problem instances correctly and is also able to reconstruct several pages at once.

# Zusammenfassung

In meiner Masterarbeit arbeite ich die Wiederherstellung von durch Shredder zerstörter Textdokumente aus. Im Gegensatz zu herkömmlicher Dokumentenwiederherstellung – die auf Farb- oder Umrissinformationen beruht – ist die Wiederherstellung von Textdokumenten noch nicht eingehend untersucht worden. Normalerweise werden Papierdokumente mittels Shredder zerstört, d.h. in längliche Papierstreifen zerlegt. In dieser Arbeit wird versucht, diesen Prozess rückgängig zu machen. Zuallererst beschreibe ich die Problemstellung formal. Als nächstes werde ich eine Möglichkeit aufzeigen wie Lösungen zu diesem Problem eingeschätzt werden können. Weiters werden eine Reihe von Verbesserungsstrategien vorgestellt, die bei der Evaluierung helfen. Definierte Konstruktionsheuristiken ermitteln gute Lösungen innerhalb kurzer Zeit. Mittels Optimierungsalgorithmen wird nun versucht eine möglichst gute Anordnung der Schnipsel zu finden, im Idealfall die ursprüngliche. Eine Testapplikation simuliert den Prozess des Shreddens einer Seite. Diese Seite wird dann mittels der oben beschriebenen Evaluierungstechniken und Optimierungsmethoden wie Multistart Variable Neighborhood Search, Simulated Annealing und Iterated Local Search wieder zusammengesetzt. Es wurden ausführliche Tests mit einem 60 Instanzen Testset durchgeführt. Die implementierte Applikation konnte mehr als die Hälfte aller Testinstanzen wieder korrekt zusammensetzen und kann auch mehrere Seiten auf einmal wiederherstellen.

## Danksagung

Ich möchte mich vor allem bei meinen Eltern Helmut und Paula Morandell bedanken, die mir in erster Linie das Studium, und damit auch diese Arbeit, ermöglicht haben.

Weiters danke ich besonders meinem Betreuer Matthias Prandtstetter, dem ich viele interessante Gespräche und viele wichtige Ideen und Verbesserungen verdanke.

Mein weiterer Dank gilt Robert Morandell und Mirja Biedermann für ihre Unterstützung und all meinen Studienkollegen, die mich durch mein Studium begleitet haben.

# Contents

# List of Figures

# 1 Introduction and motivation

The reconstruction of destroyed information on paper is of emerging interest in different areas, including the private, business and the military sector. Disposing information written down on paper is a standard process and often not done carefully enough. Skoudis [21] describes a technique known as *dumpster diving* that tries to utilize this behavior to gain access to sensitive information. People often just throw away account details or other information. It is easy to search trash and gather relevant information. There are several ways to protect oneself from these threats. A simple form of protection is to at least tear the papers before disposing them. Skoudis writes that “a well-used paper shredder” presents the best defense against dumpster diving. But there are even other applications where reconstruction may become necessary. Forensic institutions may have an interest in reconstructing paper destroyed by a presumable delinquent or even government agencies may want to recover lost information. Another great field of application is in archeology. Excavations often yield ancient artifacts that are broken or scattered [18]. These need to be reconstructed.

As one may guess there are a lot of ways to get rid of information on paper. Big institutions use so called burn bags, which are containers that eliminate paper physically e.g. with fire. But in this work the main focus is on using paper shredders which produce strip output.

Having access to shredded source material is only the first step in regaining the information. The main problem is to sort or order the bits and pieces optimally or at least semi-optimally. Doing this by hand can be tremendously time consuming or even infeasible. Computer assistance can definitely be an advantage in taking over the tedious task of trying out countless variations of piece placement.

Though there are some proprietary approaches to this problem from commercial companies this topic has not been thoroughly and lengthly examined in the academic environment.

## 1.1 Paper shredders

As we are dealing mostly with output from paper shredders I want to take a quick excursion into the world of mechanic shredders. Paper shredders come in many different flavors. Besides noise level and shredding speed the most important attribute is the output quality. Shredded material should be impossible to reconstruct. DIN[1] 32757 describes 5 different security levels for shredder output. The main difference between these security levels are their constraints put on the output. Level one just requires that processed output has a width of at most 12 mm. Normally shredders produces strips by vertically cutting pages. On the contrary level five requires the shredder to produce output that cannot be reconstructed with current state of the art. The output must have at most 0.8 mm width and 15 mm length. This is typically achieved with some sort of cross cutting. The intermediate security levels offer gradient measure of immunity from reconstruction.

Basically there are several methods for a shredder to process paper [4]:

- *Strip-cut*
  Most shredders fall into this category. This type most commonly has several rotating blades which cuts the paper vertically into rectangles.
- *Cross-cut*
  This type of shredder has two rotating drums which stamp small rectangles or diamond shaped pieces out of the input paper.
- *Other methods*
  There is a whole array of shredders which use other methods of destroying paper e.g. hammermills which press the input material through a fine screen. We will not look into these any deeper.

[1]German Institute for Standardization

## 1.2 Related Problems

There are several problems that are related to reconstructing shredded documents. In [14] Justino *et al.* describes a procedure to reconstruct documents that have been shredded by hand. Manually shredded pieces have quite different characteristics than those coming from mechanically shredded documents. First of all the cuts are apparently not parallel and at the same interval. More important two pieces torn apart by hand need not have the same edge. Paper has the awkward attribute that the edge of a torn piece may have an inner and an outer boundary. Justino proposes a polygonal approximation method to simplify the complexity of a piece. Then several features are extracted from each piece, such as angle between edges and distance between vertices. The next step is to calculate the similarity between pieces. A global search is done next. This algorithm gives good results for small instances but drops for large numbers of fragments.

Another related problem is the automated assembly of a jigsaw puzzle. Here all the pieces have almost the same surface area – almost a square – but the edges are different. For border pieces the edge is straight, all other edges have some sort of curve. So the matching algorithm has to find only a partial match between pieces. Wolfson [29] describes such curve matching techniques.

## 1.3 Outline

In this work I am going to examine the problem of simulating strip cut shredders and trying to reconstruct the resulting pieces automatically. The simulation process is relatively simple. The input consists of a picture file and is transformed into a XML file, which holds all necessary information about the snippets. This transformation tries to extract certain features from the source material. There are many different approaches to this. Ukovich [26] for example utilizes specific MPEG-7 descriptors, among others, color structure histograms or contour shapes. In our approach we examine the edges of shreds a little bit more in detail and try to find corresponding edges on other pieces. Especially with written (as opposed to pure image) documents cuts through a character leave points of color at the edges which correspond in pattern.

As already noted the input data is not scanned in shredded paper but simulated data. Therefore we neglect any image recognition problems which might arise from scanning in strips e.g. the strip is not scanned in as a straight strip but bended or added noise from the scanning process. Section 2 elaborates thoroughly on the underlying problem definition.

In section 3 I will present some related and previous work which is connected to document reconstruction. It is always an advantage to have a good understanding of similar problems and to know which approach performs good or not so good under certain circumstances.

An important step is the evaluation process. Comparing a shred with another yields a specific objective value. By optimizing the sequence of these pieces we seek out optimal results. Section 4 details the problem evaluation – how it can be done and what improvements can be added.

Optimizing the overall fitness of the document to be reconstructed is probably the most difficult aspect. Since there is a whole legion of optimization methods for such applications we will look into these a little bit further and select a promising one. This is done in section 5.

In section 6 I will present some construction heuristics which give good initial solutions in a reasonable amount of time. These may then be improved by other optimization methods.

Section 7 describes the implemented demo application. The implementation is split into three parts. First the problem is created (simulation of shredding), then this problem is solved. The result is saved in a XML file and may then be visualized on screen. The visualization of the result is important, because that is when a human viewer can determine if the result can be deciphered. For example swapping two identical or almost identical strips does not influence the result for a human viewer negatively.

# 2 Problem definition

It is essential to exactly define the problem we are dealing with. We assume that a piece of paper of rectangular shape is cut into several almost shape identical shreds. The characteristics of shredded paper strips have been researched in [4] but in this work we will focus on the following attributes:

1. there may not be an optimal unique solution
2. all shreds are produced by clear cuts
3. the orientation of each strip is known
4. the length of each strip is the same but not necessarily the width
5. strips can come from multiple pages
6. no strips are missing

The final goal is to order these strips in such a way that the original arrangement is reconstructed. First off we define the problem as *Reconstruction of Strip-Shredded Text Documents* (RSSTD). By strip-shredded we denote the shape of single strips (as opposed to e.g. manually torn paper). The term text documents differentiates between image documents which imply other characteristics than text documents, e.g. text documents only deal with binary data (background and foreground color).

It is very interesting that (even if we know the correct solution) there need not be an optimal unique solution (attribute 1). This is the case when there are identical strips (e.g. blank strips). These can be swapped without worsening or improving the solution.

Attribute 2 is necessary for the synthetic simulation of shredding. In real world examples even clear cut pages are torn at least a little bit. As soon as the shredder is older or unmaintained the blades that cut the paper get blunt and produce more fuzzy cuts. For the purpose of this work we assume that shreds fit perfectly together.

Attribute 3 mainly limits the problem. By knowing the orientation of each strip we narrow the combinations of possible solutions. That way we only look for the correct placement of each strip but not the specific orientation.

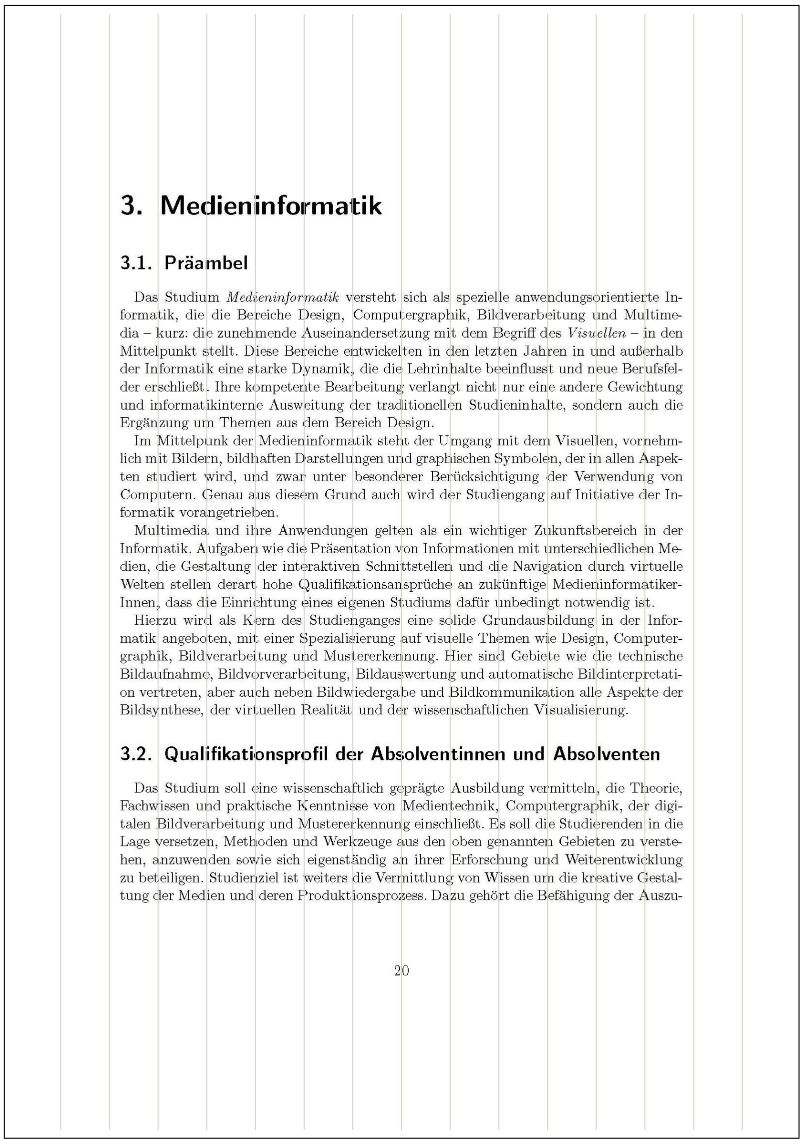
With real world data it is very unlikely that you only want to reconstruct one single page. One probably has access to a whole pile of shreds coming from multiple pages (attribute 5). Since it would be very hard to find an assignment for each strip to a page and then reconstruct it on its own, I will follow the approach to reconstruct all strips at once. It is then very easy to partition the result into multiple segments (e.g. two adjacent strips that have no pixels on their borders define a segment boundary). This can be done at the end.

Attribute 6 is quite interesting. What happens if certain strips or groups of strips are missing? In a first approach we will neglect this problem and deal with only perfect sets of input shreds.

Figure 1 shows one exemplary test page we are going to use. This is a normal A4 page with typewritten text on in. Most document pages are going to look similar. The page has been split into strips of 100 pixel each. As the page is 1600 pixel wide there are 16 strips.

Our main focus lies on reconstructing material with some sort of text on it, which means either handwritten or typewritten documents. These can be easily converted to binary data through thresholding [11]. There is quite a difference to pure image documents. Image documents consist of colored pixels. Most methods compare points at the edge of snippets with corresponding points on other snippets. The distance between these points is measured by their distance of their color values. This is for example done in [20].

In contrast to image documents written documents only consist of background and foreground, which in general can be separated by some



## 3. Medieninformatik

### 3.1. Präambel

Das Studium *Medieninformatik* versteht sich als spezielle anwendungsorientierte Informatik, die die Bereiche Design, Computergraphik, Bildverarbeitung und Multimedia – kurz: die zunehmende Auseinandersetzung mit dem Begriff des *Visuellen* – in den Mittelpunkt stellt. Diese Bereiche entwickelten in den letzten Jahren in und außerhalb der Informatik eine starke Dynamik, die die Lehrinhalte beeinflusst und neue Berufsfelder erschließt. Ihre kompetente Bearbeitung verlangt nicht nur eine andere Gewichtung und informatikinterne Ausweitung der traditionellen Studieninhalte, sondern auch die Ergänzung um Themen aus dem Bereich Design.

Im Mittelpunk der Medieninformatik steht der Umgang mit dem Visuellen, vornehmlich mit Bildern, bildhaften Darstellungen und graphischen Symbolen, der in allen Aspekten studiert wird, und zwar unter besonderer Berücksichtigung der Verwendung von Computern. Genau aus diesem Grund auch wird der Studiengang auf Initiative der Informatik vorangetrieben.

Multimedia und ihre Anwendungen gelten als ein wichtiger Zukunftsbereich in der Informatik. Aufgaben wie die Präsentation von Informationen mit unterschiedlichen Medien, die Gestaltung der interaktiven Schnittstellen und die Navigation durch virtuelle Welten stellen derart hohe Qualifikationsansprüche an zukünftige MedieninformatikerInnen, dass die Einrichtung eines eigenen Studiums dafür unbedingt notwendig ist.

Hierzu wird als Kern des Studienganges eine solide Grundausbildung in der Informatik angeboten, mit einer Spezialisierung auf visuelle Themen wie Design, Computergraphik, Bildverarbeitung und Mustererkennung. Hier sind Gebiete wie die technische Bildaufnahme, Bildvorverarbeitung, Bildauswertung und automatische Bildinterpretation vertreten, aber auch neben Bildwiedergabe und Bildkommunikation alle Aspekte der Bildsynthese, der virtuellen Realität und der wissenschaftlichen Visualisierung.

### 3.2. Qualifikationsprofil der Absolventinnen und Absolventen

Das Studium soll eine wissenschaftlich geprägte Ausbildung vermitteln, die Theorie, Fachwissen und praktische Kenntnisse von Medientechnik, Computergraphik, der digitalen Bildverarbeitung und Mustererkennung einschließt. Es soll die Studierenden in die Lage versetzen, Methoden und Werkzeuge aus den oben genannten Gebieten zu verstehen, anzuwenden sowie sich eigenständig an ihrer Erforschung und Weiterentwicklung zu beteiligen. Studienziel ist weiters die Vermittlung von Wissen um die kreative Gestaltung der Medien und deren Produktionsprozess. Dazu gehört die Befähigung der Auszu-

20

Figure 1: Test page with shredding marks

preprocessing using e.g. thresholding. We do not measure the color distance but the nearest distance between points on two adjacent edges. I will elaborate more on this in chapter 4.

## 2.1 Illustration as a graph

The problem can be formally presented as a complete asymmetric graph $G(V, E)$ consisting of vertices $V$ and edges $E$. The set of vertices $V = \{x_1, \ldots, x_n\}$ depict the single strips. The set of edges $E = \{(x, y)|x, y \in V, x \neq y\}$ are the connections between all strips. The value of these edges (defined by the function $d$) is made up by some kind of distance that still has do be defined. Since the creation of these distances is a crucial part of this work I will denote a whole chapter to it (see chapter 4).

Figure 2 shows a concrete problem instance with four strips ($V = \{1, 2, 3, 4\}$, the set $E$ of edges is made up as shown in the Figure). Sought is a hamiltonian path through the graph which yields the minimum overall distance which is subject to a certain target function (in this case this is simply the sum $\sum_{i=1}^{n-1} d(s_i, s_{i+1}) \quad s \in V$). The tour through the graph can be modeled as a permutation of $V$. For example the order $s_1 = \{3, 4, 2, 1\}$ yields a value of 43. In this case the best possible solution is $s^* = \{1, 2, 3, 4\}$ which yields 10.

Modeling problems with graphs is very common. A permutation of the set of vertices is a very natural concept to describe a tour through the graph. Optimizing this tour is often done in computer science. I will engage this subject more in chapter 5.

## 2.2 Problem complexity

When a piece of paper is shredded into $n$ pieces the solution space has $n!$ elements (since there are that many tours through the graph or permutations of $n$). Depending on the problem at hand this may even grow. Shredder remnants may be upside down if their orientation is not known which doubles

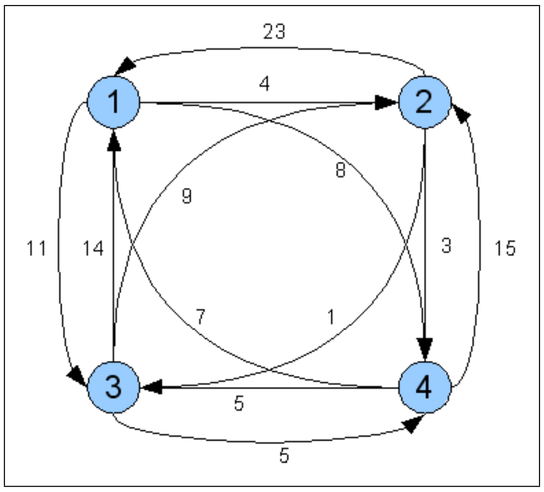


Figure 2: Problem instance with 4 strips

the solution space to $2(n!)$. If the front side is not known it is even $4(n!)$. If we generalize the problem further that each piece is a square, then it expands to $8(n!)$. Even for small problems the solution space very quickly becomes huge.

It is interesting to note that this problem definition is very similar as for the *asymmetric traveling salesman problem* (ATSP) e.g. in [7]. There a tour through an asymmetric graph is sought that minimizes the objective function $\sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(N)}, c_{\pi(1)})$. This is in fact identically to the objective function that I am going to define for the RSSTD (see chapter 4).

## 2.3 Problem boundary

During this work I will not deal with the larger parts of pattern recognition. Snippets come from the synthetic simulator which produces perfect shreds (pieces fit perfectly together). In real world scenarios this would not be the case. There is noise from the scanning process as well as noise from the shreds. If the document is cut into very slim pieces, these strips have characteristics like hair – they tend to curl. This constitutes a problem because points at the edge may not match to their corresponding point on another piece.

It is important to note that pattern recognition in general may help the reconstruction process at several places and increase quality of the result a lot. Text pages normally have a strict layout and can be processed by OCR[2] techniques very well. Through this more information about features of the problem data can be extracted. Though I will exclude this field of expertise for now.

I will also neglect page segmentation problems. Normally in real world applications a pile of strips coming from multiple pages would be available. The reconstruction process returns a sequence of these strips. Partitioning this sequence back into pages is an interesting task but not part of this work.

Another problem arises from the mechanical process of shredding. Old shredders tend to tear paper instead of cutting it, since the blades inside become blunt. That means first and foremost that information get lost, since tearing produces more powder remnant than cutting. Second the cut is not straight but jagged which will complicate the evaluation process.

## 3 Related work

Very often it is beneficial to take a look at related problems. By comparing approaches and solutions that have been established for similar problems we can reuse or adapt these methods. Since there has already been some work done with respect to the reconstruction of documents or images in general it is worth looking into them.

In [20] Skeoch does an in-depth-investigation into automated document reconstruction. Mainly the reconstruction of pure image documents is examined. This is in some terms similar to text document reconstruction. The strips are compared depending on the information on the borders. The difference here is that the strips of image documents contain much more information whereas strips of text documents mainly consist of binary

[2]optical character recognition

data (e.g. white is background, black is foreground) – the majority being background.

The main strategy is that pixels on the borders are compared to adjacent pixels on other strips' borders at the same position regarding their color values. Normally colors change smoothly in images. This behavior is used to find strips that match together very well. For this strategy several distance functions were proposed, ranging from euclidean to cosine distance. Besides this also other possible methods were mentioned like using color histograms and edge detection.

Skeoch also deals with a big part of pattern recognition. It is important to mention that this – although not the main focus of my work – has a big impact on the solution. In her work she examines the whole reconstruction process, beginning from scanned in shredded pieces. These have to be extracted. For this she explains methods for extracting rectangular and curved strips. She mentions that though both methods have their weaknesses they perform sufficiently.

Skeoch used evolutionary algorithms as optimization method to solve the problem. Single and multiple page reconstruction was tried as well as double-sided pages. Skeoch mentions that her approach worked well for synthetic data but did not scale for real life data. She also clearly notes that this approach does not perform very well for text-based images.

Ukovich *et al.* [26] follows an interesting approach to reconstruct documents. The reconstruction of strip shredded documents can be seen as a specialized form of a jigsaw puzzle. Established methods try to solve this problem by matching the curves at the edges. Ukovich now also tries to add information on the basis of the content *on* the pieces. The necessity for this is clear when the pieces' shapes are almost identical. To retrieve information about the content of pieces Ukovich uses *content-based image retrieval* (CBIR) techniques. CBIR is an emerging field of expertise [22]. There are e.g. standardized MPEG-7 descriptors [19] that have also been used in Ukovich's approach.

First of all similar strips are grouped together (e.g. a subset for all color strips, a subset for strips containing handwritten text, ...). This mainly cuts down complexity. The grouping of the shreds is done using three general features (color, texture and shape). In detail Ukovich uses three color descriptors, two texture descriptors and two shape descriptors. Ukovich also considers domain specific features like OCR and language-dependent attributes. After the grouping phase the final reconstruction is done using a complete search.

The feature selection is very important and dependent upon the type of document. Certain features work well for certain documents, others do not. Ukovich found the results to be encouraging for color images. For text documents only specific features worked limited (like using spatial color information on documents where certain lines have the same text color).

In [25] Ukovich elaborates on finding additional features apart from the already mentioned MPEG-7 descriptors. In this work notebook paper is used as source material which has slightly different characteristics than office documents e.g. different size, paper color and width. Color features can now describe the kind of paper and color ink used to segregate them.

This work also explicitly deals with handwritten text documents. Ukovich suggests the use of writer identification and handwriting classification as features but since the remnants are so small that not even one word fits on it, it is abandoned. Instead edge descriptors, like the MPEG-7 edge histogram are used.

Another very important feature which is present in all strip shredded problems is squared paper detection. When dealing with real life data, strips are usually scanned all together. They must then be separated and digitized. Finding strip patterns is not a trivial task. Ukovich uses the Hough transform [12] to recognize patterns. The Hough transform is a general purpose tool to extract features. The main advantage in using the Hough transform is that it detects any feature given in parametric form e.g. lines, curves, ellipses.

Experiments on the squared paper feature detection have been done and Ukovich reports that all remnants were detected correctly.

In another approach Ukovich [27] tries to cluster the remnants. Just in analogy with a human solving a jigsaw puzzle remnants with similar content are put together and a more intensive search can then be performed on the smaller subproblems. It is also noted that this method not only works for strip shredded problems but also jigsaw puzzle assembly and fragment reconstruction. The clustering has a twofold effect: for one it reduces the complexity of the subproblems significantly and second it improves the quality of the solution because strips are only sought in their specific clusters. This of course assumes that the classification was done correctly.

One of the integral parts of the clustering problem is to find out how many clusters there are or should be since one cannot know in advance how many pages existed. In an ideal case where all remnants are present and come from the same shredder this might be the case. But there are many possibilities that nullify this assumption. Because of that Ukovich decided to define *natural clusters*. Experiments showed that the clustering obtains good and robust results.

When comparing previous works about document reconstruction it certainly can be seen that text documents have other requirements than normal image documents regarding the reconstruction process. Methods that work well for images do not necessarily perform equally for text documents. There is definitely room for improvement here. This is exactly where I am going to hook in and try to offer some satisfying procedures to expand the field of automated reconstruction to text documents.

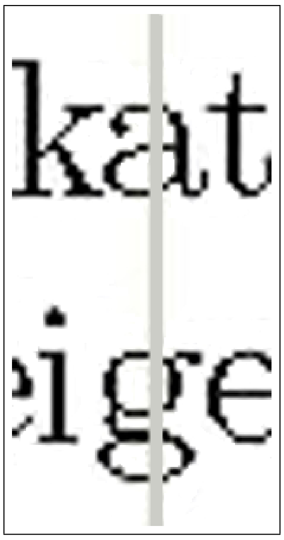


Figure 3: Close up of a cut between to strips

## 4 Problem evaluation

To generate good solutions some kind of measurement is needed, what *is* a good solution and what *is not*. If you compare a strip with two other strips, some kind of quality needs to be defined, that indicates which one suits better.

I will mainly concentrate on text documents in this work. The edges of strips are of major interest for this approach. Figure 3 shows a close up image of an edge between two characters. A character cut in half has adjacent pixels on either side of the edge. As one can see the upper character *a* is cut at three different positions. The adjacent pixels at the uppermost cut are at the same vertical level. The same applies for the pixels at the middle cut. More interesting is the bottom cut. Though the strips are correctly adjoin there is an offset between the vertical level of these pixels. If you measure the distances of these pixels on the vertical level, the sum of these distances should be as small as possible to match the strips correctly.

## 4.1 Pixel distance

Let us define the $\epsilon$-environment $U(x, \epsilon)$ of a pixel $x \in S_i$, where as $S_i$ denotes the relevant strip edge of strip $i$ ($S_i \in S$, $S$ is the set of all strips). We assume that $\epsilon$ is a non-negative value and $U(x, \epsilon)$ contains all pixels on the relevant strip edge of strip $j$ ($i \neq j$), such that the following equation holds true (the function $vert(x)$ yields the vertical position of a given pixel):

$$|vert(x) - vert(y)| \leq \epsilon \tag{1}$$

Further we assume that $d(x, y)$ denotes a function which computes a distance value of pixel $x$ and $y$ (for a detailed definition see section 4.2).

Using this assumption it is possible to introduce a function $\varphi(x, S_i)$ which computes the pixel $y \in S_i$ which is the closest pixel to $x$ on strip $i$.

$$\varphi(x, S_i) = \arg\min_{y \in S_i}(d(x, y)) \qquad S_i \in S \tag{2}$$

Next we define a function $\delta$ which returns the distance between a pixel $x$ and the closest adjacent pixel on strip $i$.

$$\delta(x, S_i) = \min_{y \in S_i}(d(x, y)) \qquad S_i \in S \tag{3}$$

## 4.2 Pixel distance evaluation

One crucial part of this thesis is the definition of a distance function $d(x, y)$, which can be used to compute good alignment estimations.

One straightforward way would be to return the vertical distances between two points. If two pixels are further apart than $\epsilon$ the distance is defined to be $\epsilon$.

$$d(x, y) = \begin{cases} |vert(x) - vert(y)| & \text{if} \quad y \in U(x, \epsilon) \\ \epsilon & \text{if} \quad y \notin U(x, \epsilon) \end{cases} \tag{4}$$

It may be beneficial to use an even graver distance measure so that pixels further apart get penalized even more. To do this, another possible method for measuring the distance, is to penalize larger distances quadratically, or to be more general to the power of $h$. Since this and the next variation of the distance function penalize pixels further apart more than the original function, I will call these modifications *heavy distance penalization.*

$$d'(x,y) = \begin{cases} |vert(x) - vert(y)|^h & \text{if} \quad y \in U(x,\epsilon) \\ \epsilon^h & \text{if} \quad y \notin U(x,\epsilon) \end{cases} \tag{5}$$

Another way to penalize distant pixels would be to punish pixels where there is no adjacent pixel in the $\epsilon$ neighborhood with a specific value $\phi$.

$$d''(x,y) = \begin{cases} |vert(x) - vert(y)| & \text{if} \quad y \in U(x,\epsilon) \\ \epsilon + \phi & \text{if} \quad y \notin U(x,\epsilon) \end{cases} \tag{6}$$

Another method to tweak the distance function is to privilege good matches. When you evaluate pixels at the strip's border and an exact match is found (i.e. there is a pixel at the exact same place of the adjacent border) this pixel is promoted by some value $\pi$ ($\pi < 0$). By doing this we favor exact matches of pixels. I will call this *exact match favoritism.*

$$d(x,y) = \pi \qquad \text{if} \quad vert(x) = vert(y) \tag{7}$$

Of course it is absolutely legal to mix these approaches.

## 4.3 Strip distance

Next we need to define the fitness $F_s$ of two strips $X$ and $Y$ – where $x \in X$ are all points on the right side of strip $X$ and $y \in Y$ are all points on the left side of strip $Y$.

$$F_s(X,Y) = \sum_{x \in X} \delta(x,Y) + \sum_{y \in Y} \delta(y,X) \tag{8}$$

$$\text{s.t.} \quad y \notin Y^* \quad \text{where} \quad Y^* = \{y | \exists x \in X : y = \varphi(x,Y)\} \tag{9}$$

The constraint $y \notin Y^*$ is necessary because otherwise we would double count some distances and thus tampering the result.

With the definition given above we first start adding up distances from left to right and then adding distances from right to left of pixels which have not already been chosen. Of course other methods of calculating the strips' fitnesses are possible. An obvious variation is to swap directions – start by adding distances from right to left and then the other way.

$$F_s'(X,Y) = \sum_{y \in Y} \delta(y,X) + \sum_{x \in X} \delta(x,Y) \tag{10}$$

$$\text{s.t.} \quad x \notin X^* \quad \text{where} \quad X^* = \{x | \exists y \in Y : x = \varphi(y,X)\} \tag{11}$$

These approaches allow a pixel to have more than one connection to an adjacent pixel. Another possible variation would be to prohibit these cases i.e. one pixel can only have at most one connection to another adjacent pixel.

## 4.4 Objective function

Having this, the fitness $F_p$ of a page $P$ is defined as the sum of the strips' fitnesses.

$$F_p(P) = \sum_{i=1}^{n-1} F_s(S_i, S_{i+1}) \qquad n = |S| \tag{12}$$

The overall goal is to find a permutation of strips which has minimal fitness.

## 4.5 Additions

It is important to mention that with the evaluation so far it can happen that evaluated solutions have better objective values than the correct solution. This for example happens when the cut is made on the side of a vertical line (so the pixels get penalized a lot). To minimize this behavior and improve the evaluation quality in general I will introduce additions to the original evaluation strategy.

### 4.5.1 Blank strip elimination

One method to reduce the problem space is to eliminate blank strips. On the test page in Figure 1 on page 7 there are four identical strips, the two on the left side and the two on the right side. These strips are completely blank, having no pixels on them at all. That also means they have no usable features for our evaluation (i.e. pixels at the border).

These blank strips are either on the border of the page or connecting two other strips. Either way they can be omitted. One has to consider that the page width will shrink when omitting blank snippets. This is particularly important when several pages are to be reconstructed – especially if the page width is used as indicator whether enough strips have been assigned to a page..

In our test application we find blank strips by looking at the strip's borders – if there are no pixels on them the strip is marked as blank and omitted. It is important to notice that there is a chance that a legitimate strip is falsely recognized as blank. This is the case when the cut at the strip's border does not intersect any letters.

Notice that these false positives happen even when each blank strip is double checked, for example by looking at all pixels of a strip. If the border of a strip is blank but not the interior we know that this is a false positive but though this does not add information to the evaluation, since we still only look at the borders. As long as the strip width is small enough the result should not be damaged too heavily.

One nice side effect of blank strip elimination is that the complexity of the problem normally gets a little easier since the number of strips decreases.

### 4.5.2 Empty border penalization

After blank strip elimination there are only two kinds of strips left: strips which have pixels on both sides and strips which have pixels on only one side (i.e. one border has no pixels). Of interest in this situation is the latter kind – I will call them *border strips* for brevity. A typical text document (see Figure 1) has two border strips, the outermost to the left ($A$) and to the right ($B$). Now, if you put $B$ beside $A$ ($B$ first, then $A$) the distance function yields 0 for this combination, meaning this is a perfect match. Since there are no pixels on either side of the borders, no meaningful distance can be defined.

When reconstructing one page, matching these border strips may not be what we want. To hamper this behavior some kind of penalization is needed otherwise these border strips would always stick together on their empty side.

A simple manner of penalizing empty borders is to assign them a specific value. Normally all distance values are known in advance. The mean value of all distances can be assigned to them to penalize empty borders.

It is important to note that a document may contain more than two legitimate border strips. If there is some kind of gap between text, which spans over the whole page (e.g. two column text), a document may have four or even much more border strips. If text starts or ends directly at the outermost part of the page there may even be only one or no border strip at all.

### 4.5.3 Limitations

Though these additions to the evaluation may improve results it may still happen that incorrect solutions are found with a better fitness. This is most often the case when cuts are at exceptional places such as near long vertical lines where the line is at one side of the cut. These incidents get penalized erroneously and worsen the result. To correct this, more features to the fitness evaluation need to be found.

## 4.6 Measuring the solution quality

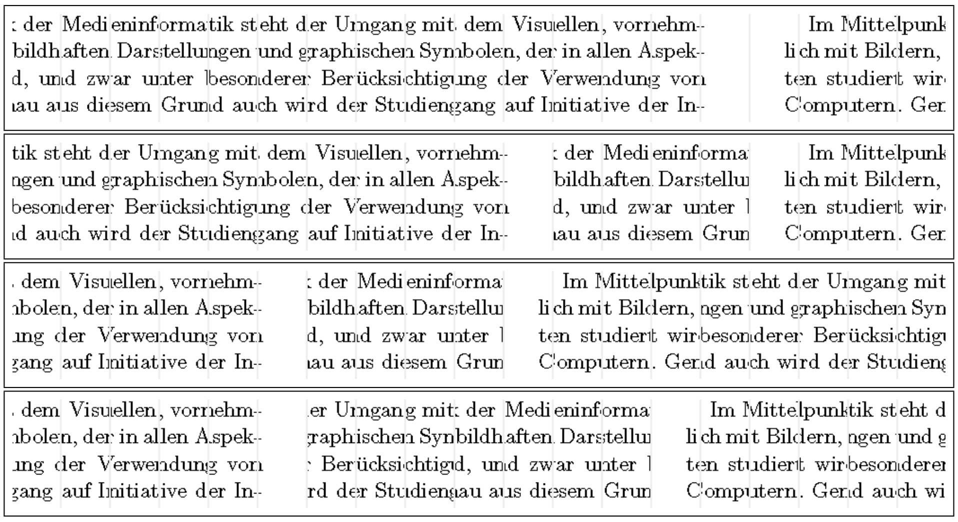
Besides the evaluation function presented earlier in this section it would be nice to have another metric which shows how good a specific solution is. By knowing the correct solution it is easy to present such a metric. When examining solutions it becomes apparent that often there are passages of correctly ordered strips (e.g. D-F-A-B-C-E-G, A-B-C being the correctly ordered passage). The longer these correctly ordered passages are the easier it is for a human reader to decipher the whole page. So it makes sense to use this attribute to define some kind of value which represents the quality of a solution.

Let $Q$ be a function which yields the number of correct sequences within a given solution. For $Q$ the following equation holds true ($s$ being a possible solution, $n$ being the dimension of the solution space):

$$1 \leq Q(s) \leq n \tag{13}$$

A value of 1 for $Q(s)$ identifies the correct solution since there is only one sequence containing all strips in correct order. On the other hand if $Q(s)$ yields the maximum value it means that no two adjacent strips are ordered correctly, meaning that a worst possible solution is found. Generally speaking the lower the value the better the solution.

With this additional metric, which I will call *sequence length quality*, it is easy to estimate the grade of a solution. A human readable solution should probably have a solution quality of at most 5.



: der Medieninformatik steht der Umgang mit dem Visuellen, vornehm- Im Mittelpunk
bildhaften Darstellungen und graphischen Symbolen, der in allen Aspek- lich mit Bildern,
d, und zwar unter besonderer Berücksichtigung der Verwendung von ten studiert wir
ıau aus diesem Grund auch wird der Studiengang auf Initiative der In- Computern. Ger

tik steht der Umgang mit dem Visuellen, vornehm- : der Medieninforma Im Mittelpunk
ngen und graphischen Symbolen, der in allen Aspek- bildhaften Darstellu lich mit Bildern,
besonderer Berücksichtigung der Verwendung von d, und zwar unter l ten studiert wir
ıd auch wird der Studiengang auf Initiative der In- ıau aus diesem Grun Computern. Ger

. dem Visuellen, vornehm- : der Medieninforma Im Mittelpunktik steht der Umgang mit
ıbolen, der in allen Aspek- bildhaften Darstellu lich mit Bildern, ngen und graphischen Syn
ıng der Verwendung von d, und zwar unter l ten studiert wirbesonderer Berücksichtigı
gang auf Initiative der In- ıau aus diesem Grun Computern. Gend auch wird der Studieng

. dem Visuellen, vornehm- er Umgang mit: der Medieninforma Im Mittelpunktik steht d
ıbolen, der in allen Aspek- graphischen Synbildhaften Darstellu lich mit Bildern, ngen und g
ıng der Verwendung von : Berücksichtigd, und zwar unter l ten studiert wirbesonderer
gang auf Initiative der In- rd der Studiengau aus diesem Grun Computern. Gend auch wi

Figure 4: Exemplary instances with quality 2, 3, 4 and 5

To compare solution qualities from different problems one can define *relative sequence length quality* $Q_r$ which also accounts for the problem size.

$$Q_r(s) = \frac{Q(s)}{|s|} \tag{14}$$

This bounds $Q_r$ to the interval $0 < Q_r \leq 1$. Getting a $Q_r$ of 1 means a total disordered solution is found. The lower the value the better the solution (the best solution would have a relative quality of $\frac{1}{|s|}$).

Figure 4 shows parts of exemplary test instances with quality 2, 3, 4 and 5. It can clearly be seen that with increasing quality the texts become harder to read.

## 4.7 Survey of the evaluation

Preliminary tests showed that with the given evaluation method plus the described additions pretty good results may be obtained. An overall score is acquired for a given solution. As the described concept tries to find adjacent pixels of side by side strips, special care must be taken to choose a reasonable value for $\epsilon$. This variable is one of the main screws of the fitness evaluation. If the value is too high or too low the results are not very good because incorrect strips get penalized either too much or too little.

Various improvements described earlier help to simplify the problem and organize the problem space. Despite all these additional evaluation updates it cannot be guaranteed that the optimal solution has the best fitness value. This turns out to be a problem because the solution space is not ordered linearly.

# 5 Solving the problem

I formulated the reconstruction of strip shredded text documents (RSSTD) as a combinatorial optimization problem like the traveling salesman problem (TSP) [1], the quadratic assignment problem (QAP) [5] or scheduling [28]. As demonstrated by Blum in [3] a *combinatorial optimization problem* (COP) can be defined by an integer set $X = \{x_1, \ldots, x_n\}$ and an objective function $f$. The set $S$ of all possible feasible assignments is called the solution space. Now one has to find the solution $s^* \in S$ where the objective value function $f$ is minimized ($f(s^*) \leq f(s), \quad \forall s \in S$).

In our case $X$ represents the set of strips. A solution to the RSSTD is a permutation of the elements in $X$. Therefore there are $|X|!$ elements in $S$. Several different ways exist to find solutions for the RSSTD. One can make an exhaustive search by so called brute-force techniques (this basically means searching through the whole problem space). Considering the magnitude of the solution space this kind of approach is most probably impractical and inefficient in general.

Generally there are exact and heuristic algorithms [3]. The former guarantee to find the optimal solution to every problem instance including a proof of optimality. Since these kind of algorithms are often not applicable for real world instances, one uses heuristics which normally return good results in reasonable time with the lack of optimality proofs.

There is a whole bunch of heuristics, starting from simple ones like local search ranging to more sophisticated methods like evolutionary algorithms or ant colony optimization. Within the next few sections I will give an overview over some of the more established (meta-)heuristics..

## 5.1 Local search

Local search (LS) [28] is a standard optimization technique. The main idea is to jump from one solution $s$ to another by inspecting its neighborhood $N(s)$. This neighborhood function should define a small set of solutions in the proximity. By consistently moving from one solution to the best solution of its neighborhood (which is also called hill climbing) the initial solution might be significantly improved.

As the name suggests, LS only uses local information about a specific solution to organize optimization. This is also its biggest drawback: when LS reaches a certain solution, which does not have a better solution in its neighborhood (a so called *local optimum*), it is trapped there and cannot improve further. It is important to note, that this local optimum need not be the globally best solution to the problem (which would be called *global optimum*). To counteract this behavior several additions to ordinary LS have been proposed e.g. iterated LS (ILS) [23]. Here as soon as a local optimum is reached the solution is disturbed in some sense. By doing this ILS tries to escape the local optimum.

Another interesting topic in LS is the choice of the initial solution, which is the starting point of the optimization. One can use a construction algorithm to generate a good initial solution. But to cover a broad area of the solution space it may also be beneficial to start from random solutions. LS can also perfectly be combined with other optimization methods. Incumbents found by them can then be locally improved.

## 5.2 Variable neighborhood search

Neighborhood searches (NS) in general rely on the fact that given a valid solution $x$ to the problem one can define a neighborhood $N(x)$ which consists of all valid solutions reached by applying one or several predefined moves. Such a move defines, how new solutions can be derived from $x$ by simple operations like swapping two features of a solution.

Sometimes it is easy to define more than one move and therefore several neighborhood structures $N_1, N_2, \ldots, N_k$ are implied by those $k$ moves. It is obvious to use a search method that benefits from all these different neighborhood structures. Variable neighborhood descent (VND) is exactly such a local search method, which tries to find a local optimum in respect to all defined neighborhood structures $N_1, N_2, \ldots, N_k$.

This can be done by examining one neighborhood structure as long as some improvements can be achieved. If a local optimum is reached, the optimization process continues to examine the next neighborhood. As soon as no improvement can be achieved in the second neighborhood the search continues with the next neighborhood. This procedure is repeated until there is no further improvement possible in any neighborhood.

Unfortunately, even this procedure can get stuck in local optima which might happen, if not all theoretical possible neighborhoods are defined (and searched). To esacpe these local optima variable neighborhood search (VNS) implements a perturbation procedure in VNS such that each time a local optimum is reached within VND, further random solutions are produced to broaden the search. Depending on the integration of VND-like searches in VNS, there are several different variations of VNS like reduced VNS (RVNS), skewed VNS (SVNS) and variable neighborhood decomposition search (VNDS). A very good reference on VNS in general and the most common variations is [17] and [13].

## 5.3 Simulated annealing

Simulated annealing (SA) [15] is a variation of LS invented in 1983. To avoid getting stuck in local optima SA also allows jumps to worse solutions instead of only moving to better ones. This is a stochastic process. The chance that worse solutions are allowed is high at the beginning and is decreased during the optimization process.

The probability when worse solutions are accepted is controlled by the so called *cooling function*. This function governs diversification and intensification of the optimization. If the cooling happens too fast SA gets trapped too early in some local optimum. On the other hand if the schedule is too slow SA keeps accepting worse solutions and not settling down.

SA resembles the annealing of metal, which descends into a low energy configuration. By doing this correctly the metal does not exhibit any cracks or bubbles. SA is one of the first metaheuristics ever invented.

## 5.4 Tabu search

Tabu search (TS) [10] is a very popular heuristic for solving combinatorial problems. The main idea is to maintain a memory of already visited solutions and add them to the tabu list. That way the algorithm avoids getting stuck in local optima and implements an explorative strategy. The size of the tabu list (tabu tenure) controls the optimization process – small values explore the near neighborhood of a given solution, big values explore larger regions of the solution space.

During time several improvements were introduced to the originally proposed approach. For instance Taillard [24] presented a method where the tabu tenure is periodically changed randomly within a predefined interval. A more dynamic handling of the tabu tenure was presented by Battiti [2]. The tabu tenure is increased if higher diversification is needed (e.g. if repetitions are recognized) and decreased if intensification is needed (e.g. if no improvements are found for some time). Another proposal does not hold whole solutions in the tabu list [3]. Maintaining complete sets of solutions in memory is highly inefficient. Therefore only so called attributes to solutions are saved and compared. Attributes are features of solutions e.g. differences between two solutions. Of course some information get lost because attributes cannot represent whole solutions.

## 5.5 Evolutionary computation

Evolutionary computation (EC) tries to incorporate nature's principle of the *survival of the fittest*. The concept is to maintain a population of solutions (called individuals). On this population the operations *recombination*, *mutation* and *selection* are performed. The main idea is to mimic nature's capability to adjust itself to changing environmental properties.

The field of evolutionary computation is huge as there exist several modifications like evolutionary programming, evolutionary strategies and genetic algorithms. The readers is directed to [16] for a deeper insight.

## 5.6 Ant colony optimization

Ant colony optimization (ACO) [9] is another nature inspired algorithm. Real ants find shortest paths between food and their nest by placing pheromones while they walk. These pheromones are then recovered by other ants. This resembles a parametrized probabilistic model.

Artificial ants start walking randomly in a completely connected graph, where its vertices are the solutions. When an artificial ant finds pheromones on edges the probability the ant follows an edge is calculated by the amount of pheromones found.

# 6 Construction heuristics

Some optimization algorithms need an initial solution to start from (or it may be at least beneficial). Simple iterated local search for example starts each time it reaches a local optimum from a new – mostly randomly chosen – solution. In our case a randomly generated solution would be produced by merging strips by chance. But under some circumstances this may not be a very promising way (e.g. if the problem space is very big). Because construction heuristics have proven to be successful in other areas (e.g. see the Christofides heuristic for the TSP in [6]) it makes sense to also define construction heuristics for the RSSTD. The goal is to generate good solutions very fast. In the following I will present some possible construction heuristics for the RSSTD.

## 6.1 Forward page construction

The main idea of this construction heuristic is to greedily reconstruct the page from left to right by first randomly choosing one strip (possibly with one empty side on the left) and subsequently adding strips to the right by choosing the best fitting one. If two or more strips yield the same result if added the first such found strip is chosen. Pseudocode for it is given in Figure 5.

One strip is chosen and each of the remaining $n-1$ strips is appended at every iteration once (there are $n-1$ such iterations). Considering that fitness calculations in this case can be done in constant time (this is described in more detail in chapter 7.2) the whole algorithm has $O(n^2)$ time complexity.

## 6.2 Duplex page construction

This is a slight modification to forward page construction. Again at the beginning a random strip is chosen. Then the best matching strip is sought

```
init x as empty solution
list = create list of all strips
remove random strip from list and add to x
while there are strips in list
  init tfitness
  for all s in list
    add s to x
    if fitness(x) < tfitness
      s' = s
      tfitness = fitness(x)
    remove s from x
  remove s' from list and add to x
return x
```

Figure 5: Pseudocode of forward page construction

and appended by matching it on both ends to the left and to the right. Also here in case of a tie between strips simply the first best matching strip is chosen. In case of a tie between sides (on the left or the right side of the sequence) the right side is chosen. The aim of trying to match the strip on both ends is to improve the result further. Figure 6 shows a pseudocode implementation.

Compared to the before mentioned forward page construction here the single strips are appended twice, to find out at which position they fit best. So this adds a constant factor to the complexity but does not change the overall complexity - duplex page construction also has $O(n^2)$ complexity.

## 6.3 Randomized duplex page construction

Some heuristics need to start from several completely different points in the solution space. A valid random solution can be chosen as starting point, but this is like looking for the needle in the haystack. It is better to start from

```
init x as empty solution
list = create list of all strips
remove random strip from list and add to x
while there are strips in list
  init tfitness, addposition
  for all s in list
    add s to x on the right side
    rfitness = fitness(x)
    remove s from x
    add s to x on the left side
    lfitness = fitness(x)
    remove s from x
    if min(lfitness, rfitness) < tfitness
      if lfitness < rfitness
        addposition = left
      else
        addposition = right
      s' = s
      tfitness = min(lfitness, rfitness)
  remove s' from list
  if addposition == left
    add s' to x on the left side
  if addposition == right
    add s' to x on the right side
return x
```

Figure 6: Pseudocode of duplex page construction

relatively good solutions. Forward and duplex page construction start with a random strip and append best matching strips accordingly. When there are $n$ strips there are at most $n$ different solutions these heuristics can deliver.

Randomized duplex page construction starts with a single snippet by chance. From the remaining strips a random one is added either to the left or to the right – wherever it matches best. In case of a tie on both ends the strip is appended on the right side. Solutions created in such a way are normally worse then from forward and duplex page construction but the space of producible solutions is much broader. The pseudocode for it is given in Figure 7. Contrary to forward and duplex page construction in randomized duplex page construction each strip is instantly added to the final solution. So the complexity is linearly dependent of the number of strips, i.e. it has $O(n)$ complexity.

# 7 Implementation

The described evaluation and solution methods have also been implemented in a demo application. For the sake of compatibility and portability several XML formats have been defined. So each step can be handled by different applications (e.g. on different systems). During the whole reconstruction process there are multiple steps that have to be done, each consisting of several subtasks.

1. Preparation
    - read in image, threshold, and segment into strips
    - write formalized problem data into XML file
2. Solution
    - read in problem data from XML file
    - precalculate the fitness matrix

```
init x as empty solution
list = create list of all strips
remove random strip from list and add to x
while there are strips in list
  s = random strip of list
  remove s from list
  add s to x on the left side
  lfitness = fitness(x)
  remove s from x
  add s to x on the right side
  rfitness = fitness(x)
  remove s from x
  if lfitness < rfitness
    add s to x on the left side
  else
    add s to x on the right side
return x
```

Figure 7: Pseudocode of randomized duplex page construction

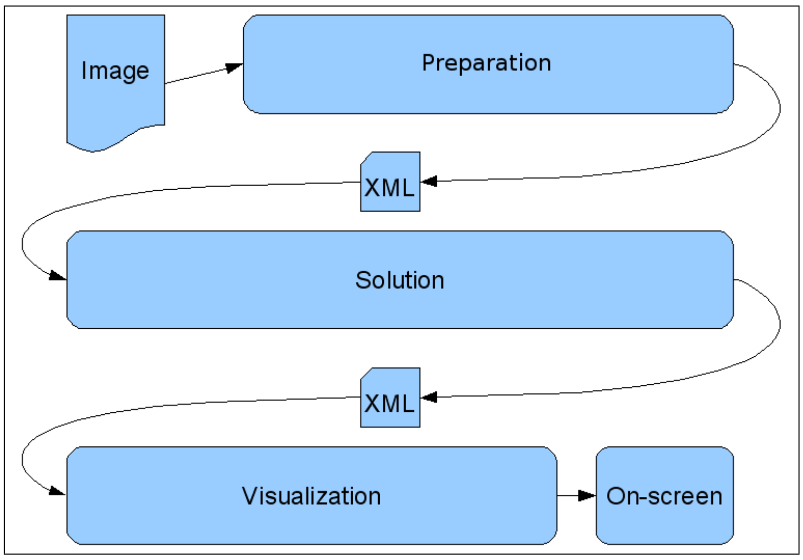


Figure 8: Workflow of the demo application

- solve the problem
- write solution into XML file

3. Visualization
    - read solution from XML file
    - visualize solution

Each of the main tasks can be done independently. Figure 8 shows the overall workflow. Since it is often the case that special solution methods are only available for certain operating systems or programming languages, the XML interface makes it easy to communicate between different systems. In the following I will go into more detail for each of these steps.

The demo application has been implemented in Java having around 3000 lines of code. The framework uses the object oriented paradigm, meaning that for example each optimization algorithm is realized as class. More algorithms can easily be added by subclassing the superclass. XML handling (reading and writing XML files) is done by Java's SAX facilities. Care has also been taken to use newer Java features like generics and enumerated types. To be useful in a scripted environment a CLI (command line interface) client has been written (instead of a graphical user interface). A GUI client can easily be created later on if needed.

## 7.1 Preparation

The preparation stage performs all tasks necessary to create a valid problem instance. Since using random data (i.e. a page with random pixels on it) is not very significant, it is better to use real life data. You feed an image (best is a monochrome text image) to the application. Besides that also the strip width is important. Let $w_i$ be the image width and $w_s$ be the strip width in pixels, then the input image is transformed into $\lceil \frac{w_i}{w_s} \rceil$ strips, each having a left and right border. For each border the coordinates of the pixels is recorded and saved into an XML file together with additional data like strip width and image width.

## 7.2 Solution

Solving the problem is probably the most interesting phase. Several algorithms have been implemented, including iterated local search, simulated annealing and a multistart VND. The test results are described in more detail in chapter 8. In the following I will present information about the used solution representation, moves used for defining neighborhood structures and implemented meta-heuristics.

### 7.2.1 Solution representation

A solution $S$ is represented formally as a sequence of $n$ input variables $S =< s_1, s_2, \ldots, s_{n-1}, s_n >$ that are stored in an array. Any such sequence of $S$ will be denoted by $\sigma_i$ in the following if this improves readability ($\sigma_i$

can also denote an empty sequence). Each entry in the array represents a certain strip, respectively contains an unique strip identification number. So to speak an element in the array implicitly shows its distinct position in the solution vector.

For the data structure which holds solutions Java's `ArrayList` data type is used. As mentioned in Java's documentation add-operations take amortized constant time. That means adding $n$ elements takes $O(n)$ time. Removing an object can be done in linear time, whereas retrieving an item is performed in constant time.

For increased efficiency, an incremental objective function update is implemented for all operations on solutions. Therefore the fitness of a solution is stored instantly as it is calculated. For that we have to sum up the fitness values for each pair of strips placed next to each other. This clearly can be done in linear time. As soon as a new solution is derived no complete recalculation is needed. Pseudocode for the incremental fitness update is given in Figure 9. By only evaluating the fitness values adjacent to the block of strips that is being moved (if the block has length one only a single strip is moved) and updating the known total fitness of the solution, the fitness of the derived solution can be decided by at most six calculations, which means this can be done in constant time.

### 7.2.2 Insertion moves

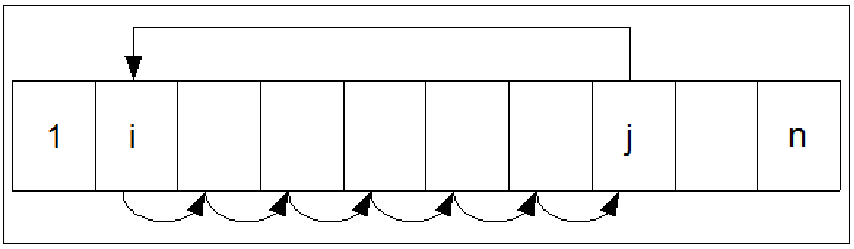
Insertion moves are defined by picking a specific strip and inserting it at another location. Because of the chosen solution representation a certain number of additional strips have to be moved to make place for the inserted strip. Due to these additional moves this operation can only be done in linear time. The worst case scenario would be to move a strip from one side of the solution to the opposite. In this case all other strips also have to be moved.

Formally insertion moves can be defined as a function

$$f_I(<\sigma_1, \sigma_i, \sigma_2, \sigma_j, \sigma_3>) =< \sigma_1, \sigma_j, \sigma_i, \sigma_2, \sigma_3 >$$

```
move_strip(solution, source, length, dest) {
  fit = fitness(solution)
  if has_left_neighbor(source)
    fit = fit - fitness(source-1, source)
  if has_right_neighbor(source+length-1)
    fit = fit - fitness(source+length-1, source+length)
  if has_left_neighbor(source)
    and has_right_neighbor(source+length-1)
    fit = fit + fitness(source-1, source+length)
  x = solution[source] to solution[source+length-1]
  remove x from solution
  insert x at position dest
  if has_left_neighbor(dest)
    fit = fit + fitness(dest-1, dest)
  if has_right_neighbor(dest+length-1)
    fit = fit + fitness(dest+length-1, dest+length)
  if has_left_neighbor(dest)
    and has_right_neighbor(dest+length-1)
    fit = fit - fitness(dest-1, dest+length)
  return fit
}
```

Figure 9: Pseudocode for fitness updates



```
insertion_move(solution, source, dest) {
  if dest < source
    offset = 1
  else
    offset = -1
  tmp = solution[source]
  for i = source-offset to dest
    solution[i+offset] = solution[i]
  solution[dest] = tmp
  return solution
}
```

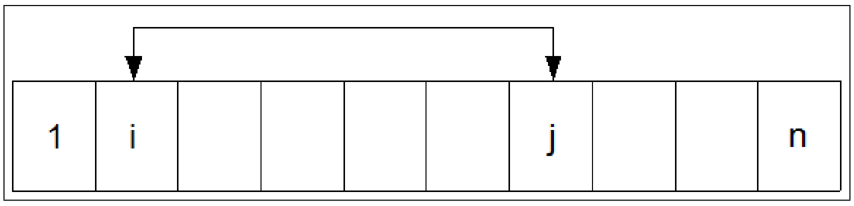
Figure 10: Pseudocode for an insertion move

which inserts the strip from position $j$ at position $i$ ($i \neq j$, $|\sigma_j| = 1$). Figure 10 shows pseudocode and a graphical representation for this move.

Due to the fact that each strip can be moved to all other possible positions a neighborhood defined by insertion moves consists of $n^2 - n$ different solutions. Based on the chosen solution representation evaluating such a neighborhood is in the time complexity of $O(n^3)$.

### 7.2.3 Swap moves

A swap move denotes the simplest move that can be performed based on our solution representation. Two strips are selected and swapped with each other. All other strips remain at their original position. Using an incremental



```
swap_move(solution, source, dest) {
  tmp = solution[source]
  solution[source] = solution[dest]
  solution[dest] = tmp
  return solution
}
```

Figure 11: Pseudocode for a swap move

fitness update function this move can be performed in constant time since the swapping itself is also a constant time operation.

Based on our solution representation swap moves can formally be defined as a function

$$f_S(<\sigma_1, \sigma_i, \sigma_2, \sigma_j, \sigma_3>) =<\sigma_1, \sigma_j, \sigma_2, \sigma_i, \sigma_3>$$

which swaps two strips $i$ and $j$ ($i \neq j$, $|\sigma_i| = 1$, $|\sigma_j| = 1$). Figure 11 shows a graphical representation of this move together with pseudocode.

Since every strip can be swapped with all other strips, a neighborhood based upon swap moves can easily be defined. Just like with insertion moves it consists of $n^2 - n$ different solutions. However since a swap can be done in constant time contrary to insertion this neighborhood can be evaluated in $O(n^2)$ time complexity.

### 7.2.4 Insertion block moves

Insertion block moves are an extension to regular insertion moves. Here not a single strip but at block of random length is selected and inserted at another location. This block can be as short as two strips and as long as $n-2$ strips ($n$ being the total number of strips). A block of length one or $n-1$ would reduce this move to regular insertion and length $n$ would render this move useless since that block cannot be moved. Analogue to regular insertion, inserting a block at another location moves additional strips to make place.

Formally insertion block moves can be defined as function

$$f_{IB}(<\sigma_1, \sigma_i, \sigma_2, \sigma_j, \sigma_3>) = <\sigma_1, \sigma_j, \sigma_i, \sigma_2, \sigma_3>$$

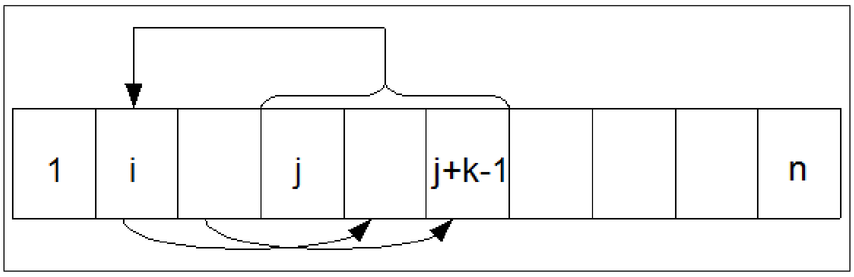
which inserts $k$ strips from position $j$ at position $i$ ($i \neq j$, $|\sigma_j| = k$, $k > 1$).

A neighborhood defined on insertion block moves tries to insert a block at all possible locations. Because additional strips have to be moved this operation takes linear time. The implementation does not examine the whole neighborhood with varying block length but chooses a certain random block length while dwelling in that neighborhood. Due to this the neighborhood has $O(n^2)$ different solutions. Similar to regular insertion this move has $O(n^3)$ time complexity. Pseudocode for this move is given in Figure 12.

### 7.2.5 Multistart VND

In this thesis a slight modification of the standard VND is used, a so called multistart VND. Just like standard VND, the multistart VND is based on different neighborhood structures, which are systematically searched. But contrary to standard VND the search procedure is restarted as soon as no further improvements can be found. The neighborhood structures used are based on the previously defined moves: insertion, swap and block insertion.

The multistart VND uses a total of three neighborhoods. The first neighborhood is defined by insertion moves. This means strips are systematically inserted at other locations in random order so long as an improvement can be achieved. The second neighborhood is based upon swap moves. Two random strips are swapped, using the same characteristic as the first move. The next and last neighborhood is defined by block insertion



```
insertion_blockmove(solution, source, dest, length) {
  if dest < source
    offset = 1
  else
    offset = 0
    dest = dest+length-1
  for i=0 to length-1
    insertion(solution, source+(i*offset), dest+(i*offset))
  return solution
}
```

Figure 12: Pseudocode for an insertion block move

```
init best_solution
do
  x = initial solution
  k = 1
  while k <= 3
    x’ = first($N_k$(x))
    if fitness(x’) < fitness(x)
      x = x’
      k = 1
    else
      k = k+1
  if fitness(x) < fitness(best_solution)
    best_solution = x
until termination condition == true
return best_solution
```

Figure 13: Pseudocode of the multistart VND

moves. A random sequence of strips is selected and inserted at every possible location. All neighborhoods are examined using a first improvement step function. As soon as no improvements in one neighborhood can be found any more, the next neighborhood is evaluated. But if a solution could be improved the process restarts from the first neighborhood. At the end the so found solution is optimal regarding to all neighborhoods. When the last neighborhood is in an optimum a new solution is generated and the algorithm restarts.

This behavior is also described with pseudocode in Figure 13. The variable *best_solution* always contains the best solution found so far. At the end this value is returned. The neighborhoods $N_1$, $N_2$ and $N_3$ are defined on insertion, swap and block insertion moves. Furthermore the function `first` retrieves the first improvement of a solution in the neighborhood $N_k$.

```
0: init best_solution
1: x = initial solution
2: x' = N(x)
3: if x' < x then
4:   x = x'
5:   goto (2)
6: best_solution = x
7: if termination condition == false
8:   goto (1)
9: return best_solution
```

Figure 14: Pseudocode of iterated local search

#### 7.2.6 Iterated local search optimization

Additionally a simple iterated local search strategy has also been implemented. A single neighborhood is chosen and searched as long as improvements are found. If no improvement can be achieved anymore the algorithm either restarts or terminates depending on a certain termination condition (e.g. time constrain). The overall best solution found is always saved and returned in the end. As possible neighborhood structures the same as for the above mentioned VND are used. The general outline of the algorithm is shown in Figure 14.

#### 7.2.7 Simulated annealing optimization

Also a simulated annealing algorithm has been implemented. The pseudocode for it is displayed in Figure 15. A single neighborhood is chosen at the beginning and then searched. The possible neighborhoods are the same as with the VND. The implementation in this thesis uses geometric cooling of the synthetic temperature. For this the temperature parameter $T$ needs to be initialized to $f_{max} - f_{min}$. Since both values are unknown, upper and lower

```
T = $f_{max} - f_{min}$
x = initial solution
repeat
  x' = N(x)
  if x' < x then
    x = x'
  else
    Z = random(0, 1)
    if Z < $e^{(-|f(x')-f(x)|/T)}$ then
      x = x'
    T = T $\cdot \alpha$
until termination condition == true
```

Figure 15: Pseudocode of simulated annealing

bounds are used. Because we can compute all strip distances beforehand we can define such bounds knowing the maximum and minimum strip distance. Further a cooling factor $\alpha$ must be chosen. This factor determines how fast the temperature cools down and such how long worse solutions are accepted probabilistically. In this thesis a high cooling factor of 0.999 is used by default (and such a slow cooling process). Preliminary tests suggested that this value works reasonable. At the beginning $T$ is high and many worse solutions are accepted meaning that a large part of the solution space is accessible. During time $T$ drops and worse solutions are accepted more improbable.

### 7.2.8 Exhaustive search

As already mentioned it can be the case that the optimal solution may not have the best fitness value. To find the best fitness for a specific problem a complete search can be done (this of course is only practicable for relatively small instances). Therefore an exhaustive search has been implemented, which means a complete enumeration of all possible solutions is done. This is

```
next() {
  i = N - 1
  while (value[i-1] >= value[i]) do
    i = i - 1
  j = N
  while (value[j-1] <= value[i-1]) do
    j = j - 1
  swap(value[i-1], value[j-1])
  i = i + 1
  j = N
  while (i < j) do
    swap(value[i-1], value[j-1])
    i = i + 1
    j = j - 1
}
```

Figure 16: Pseudocode for exhaustive search

achieved by evaluating all permutations of the problem vector. The algorithm to create the necessary permutations is given in Figure 16 and described in more detail by Dijkstra in [8]. The solution is at first stored in ascending order in array `value` and `N` depicts the size of the array. Each call to `next()` gives the next permutation of the vector.

## 7.3 Visualization

The last part is to visualize the found solution. This is very important to evaluate the reconstructed page from a human point of view. Only so the real quality of a solution can be perceived. Since blank strip elimination has been performed on the input data, the visualized resulting page is probably going to be slimmer.

The visualization is mostly done by Java's `imageIO` package. Figure 17 shows a sample solution for the P2 test instance (in fact this solution has a quality of 3, meaning there are three correctly aligned consecutive sequences of strips).

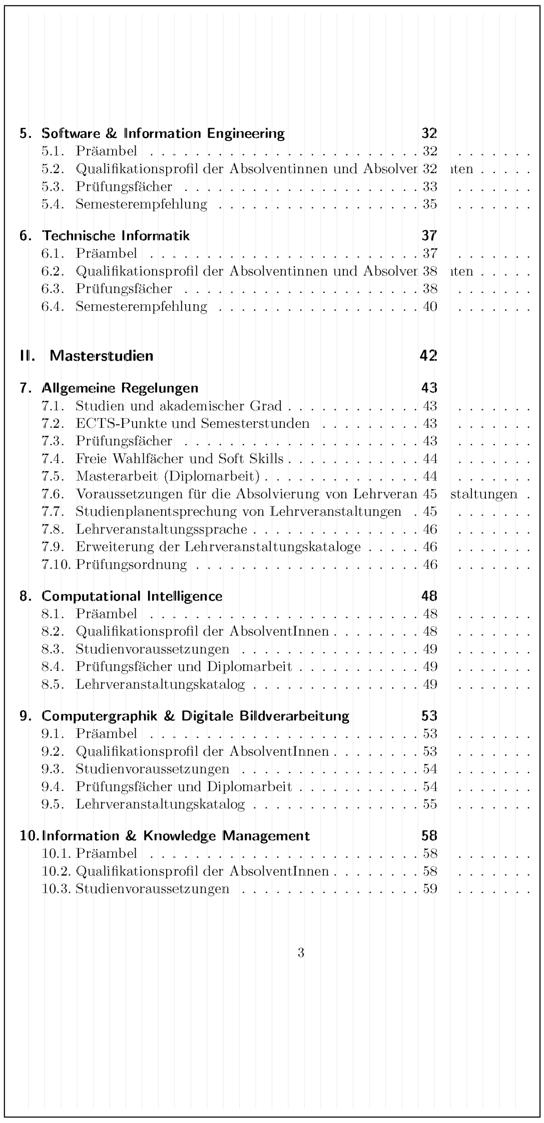
# 8 Tests

For testing purpose several problem instances were created from a typical typewritten A4 page document (see appendix D, which displays all used test images). These instances offer a general overview of the effectiveness of the evaluation and solution methods. The complexity increases steeply for the test instances by increasing the resolution and decreasing the strip width. The exact data of the test instances can be found in appendix A.

The test pages were scanned with several resolutions (72, 150, 300 and 600 dpi) and stripped at different strip-widths (20, 50 and 100 pixel strip width). The resolution is important to the reconstruction process since having a higher resolution makes it more improbable that cuts are at unfavorable positions and there is more data that can correlate. This in turn enables the evaluation to produce good results.

For $\epsilon$ the value 10 was taken. The parameter $\epsilon$ describes the distance within pixels on another strip are looked for. Empty border penalization, which penalizes borders without any pixels at all, was set to the average of all evaluated objective values. Heavy distance penalization, which also increases penalization, was set to quadratic. Exact match favoritism, which favors sets of pixels which match in their vertical level perfectly, was set to $-50$. For measuring strip distances the normal *left-right* type was chosen. As construction heuristic duplex page construction was selected. Preliminary tests revealed that this is a set of parameters which works reasonable well.

Several test instances have been run twice using both strip distance calculation methods (see chapter 4.3). These samples have been compared using the statistical unpaired two-tailed t-test. The tests were made using the same setup as mentioned in the previous paragraph. It was found out



3

Figure 17: Sample solution for the P2 test instance

that the difference of the resulting quality of the solutions can be considered to be not statistically significant. To be consistent in all other tests the first mentioned calculation method (left-right) was used.

A comparison of the test results made in this thesis to other reconstruction methods is only limited possible. For once the application to text documents is not as frequent as image data. For example Skeoch explicitly mentions that her approach does not work as well for text data as for image data but no specific results are mentioned. She also for the most part uses real life data from scanned source material. This changes the problem (respectively the produced result) significantly because with simulated data a perfect solution is at least possible whereas with real life data one normally works with approximations. On the other hand Ukovich mainly tries to cluster shredded remnants which is a different topic.

## 8.1 Evaluation of the construction heuristics

The construction heuristics were tested independently. The results are recorded in appendix B. Both forward page construction (FPC) and duplex page construction (DPC) perform mostly good – in most instances DPC being better. Construction heuristics play a big part in the reconstruction process. For the most part they offer very good approximations which can then be improved further.

Figure 18 and 19 show two typical test results for the construction heuristics. In the Figures *quality* means average quality over all runs and $SW$ means strip width. The first Figure shows results for the test instance P1 (with 150 dpi), the second for test instance P5 (with 600 dpi).

As can be seen from the Figures DPC yields for the most part better results then FPC. RDPC scales worst from all construction heuristics. DPC and FPC perform much better for bigger instances. Over all runs DPC could outperform FPC 15 to 8 times, meaning DPC found a better result quality.

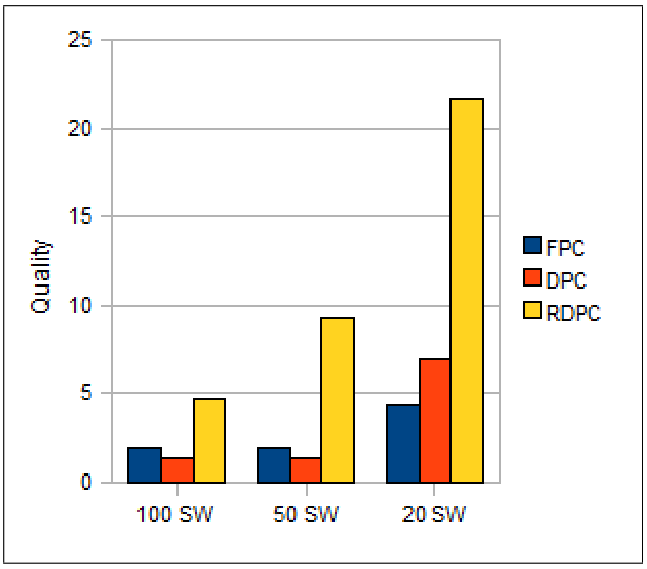


Figure 18: Construction heuristic results for P1 with 150 dpi

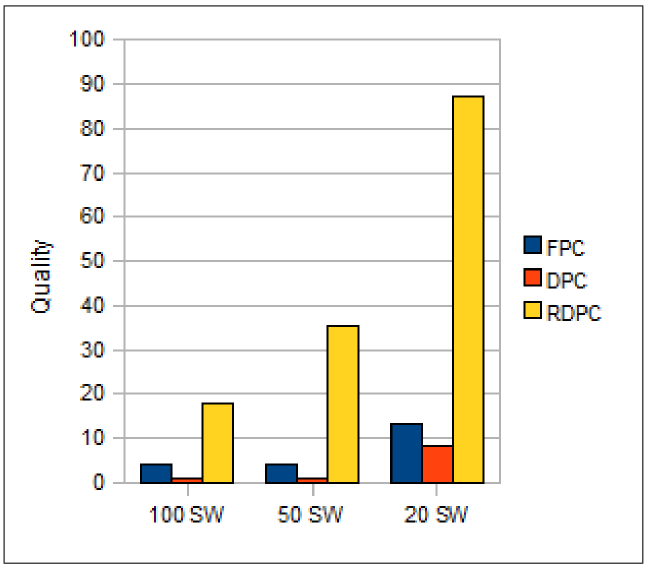


Figure 19: Construction heuristic results for P5 with 600 dpi

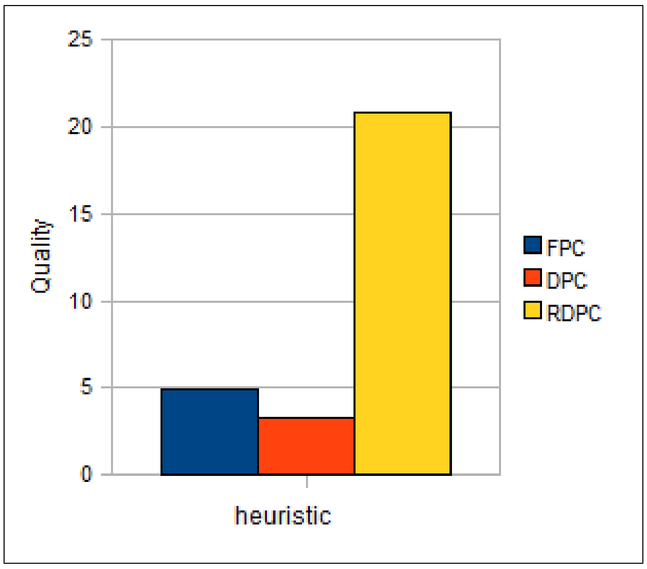


Figure 20: Average results for the construction heuristics

Looking upon average solution quality DPC yielded 55 to 5 times better results.

To back up this individual data I want to present two more results showing the average of the construction heuristic data. Figure 20 shows the average results for each construction heuristic. The shown results resemble very much the individual data which means we really can assume that on average DPC should yield the best results. Whereas Figure 21 shows the average results per page. As we will see in the next chapter the average page results also resemble the best found solutions after optimization (e.g. test page P2 is the hardest page to solve, which is also visible from the Figure). The averages in this latter Figure are only based on FPC and DPC.

## 8.2 Evaluation of the optimization

As already mentioned, several optimization methods were implemented. As main optimization method the multistart VND was chosen but also the other methods are discussed. The detailed test results for the VND are presented

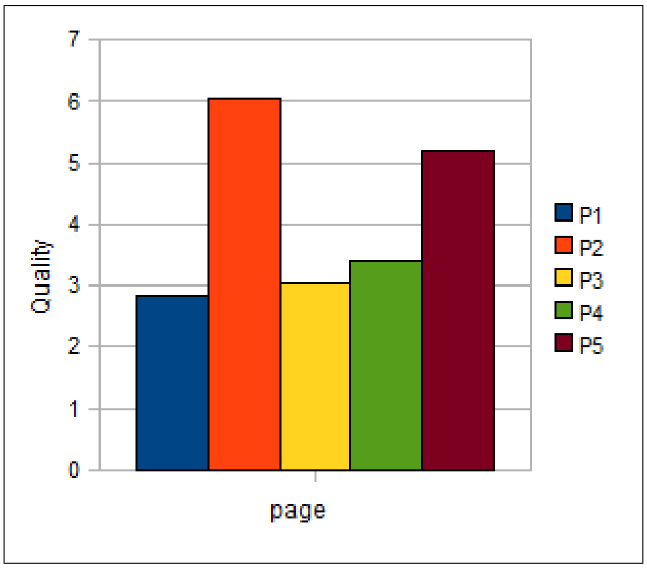


Figure 21: Average results for the construction heuristics for each page

in appendix C.

As can been seen the instances with higher resolutions tend to be easier to solve. Five 72 dpi instances (all strip widths included) have been solved optimally where as ten 600 dpi instances with the same strip widths have been solved with perfect quality. In detail 33 out of the 60 test instances have been solved optimally, 23 have been solved with a quality between two and five, that leaves four instances above a quality of five.

The easiest page to solve is page P3 (all instances solved optimally), then page P1 (11 of 12 solved optimally), then page P5 (7 of 12), then page P4 (2 of 12) and the hardest is P2 (1 of 12). The pages P1 and P3 are quite similar, there is much text on it and some headlines. These patterns tend to be easily reconstructible. P4 and P5 are also quite similar, P5 having a table in it. The reconstruction method seems to handle graphic objects quite good. That is probably why P5 is easier to solve than P4. P2 is the hardest page to solve because there are many patterns that repeat themselves. Since the reconstruction strategy tries to match borders it cannot handle many equal looking strips very good. Two typical examples how mismatching may look

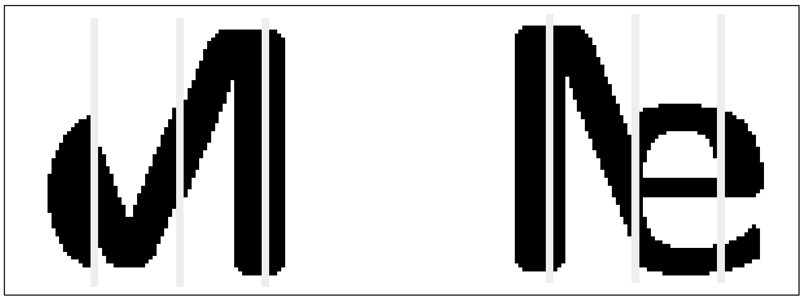


Figure 22: Typical mismatches

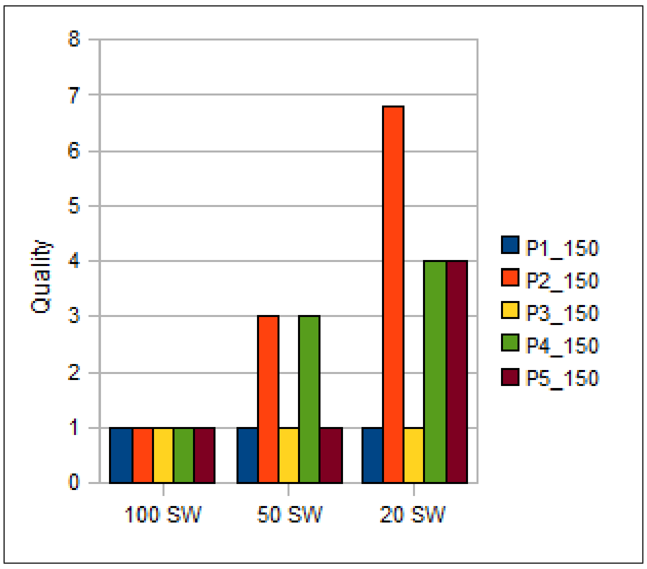


Figure 23: Optimization results for all instances with 150 dpi

like are shown in Figure 22.

The Figures 23 and 24 show the summarized results of all the test instances with 150 and 600 dpi. *Quality* means average quality over 30 runs and *SW* means strip width. Is can be seen P1 and P3 are always solved correctly. P2 becomes with decreasing strip width much harder to solve in the 150 and the 600 dpi case. It also can be seen that P4 and P5 profit from the higher resolution and yield better results. P5 can even be solved optimally with 600 dpi and 20 pixel strip width.

In general problems become harder to solve with decreasing strip width.

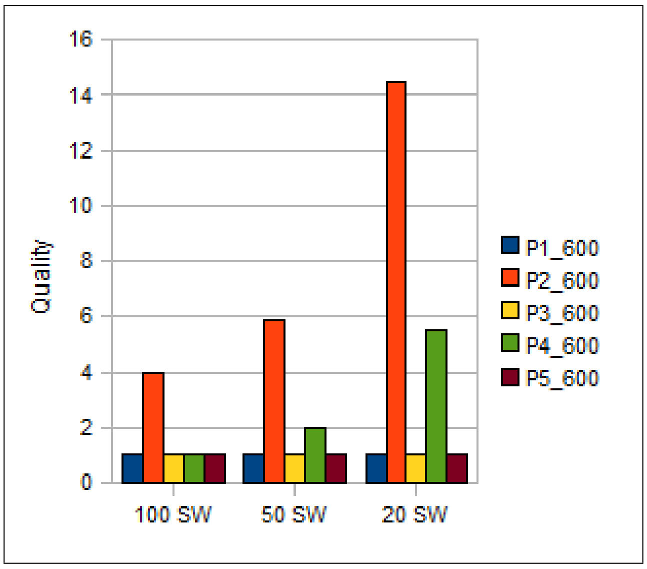


Figure 24: Optimization results for all instances with 600 dpi

Primarily of course because the solution space grows. This is shown graphically in Figure 25 which shows the average results of each strip width. Finally average results for all test pages are displayed in Figure 26 which summarize the before mentioned result order P3, P1, P5, P4 and P2 (from best to worst).

## 8.3 Investigation of $\epsilon$

As already mentioned the parameter $\epsilon$ (which defines the area where adjacent pixel are sought) plays a central part in the evaluation. Having small $\epsilon$ gives little penalization for pixels which have no adjacent partner. To view the consequences of changing $\epsilon$ several tests have been run with all test instances using several different $\epsilon$ values (2, 5, 10, 20, 40 and 100). As parameter setup the configuration at the beginning of this chapter is mentioned.

Figure 27 and 28 show the results. The vertical axis gives the result quality averaged over 30 runs, the horizontal axis shows the used $\epsilon$ value. The former graph shows the results for test instance P4 with 600 dpi and 20 pixel strip

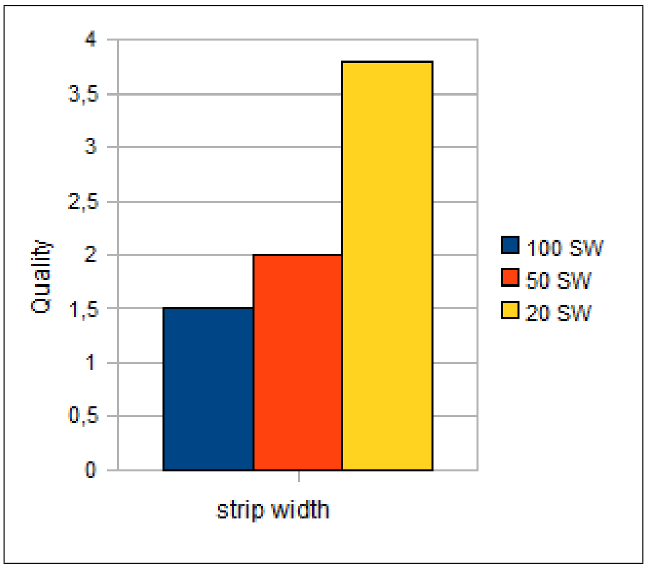


Figure 25: Average optimization results regarding strip width

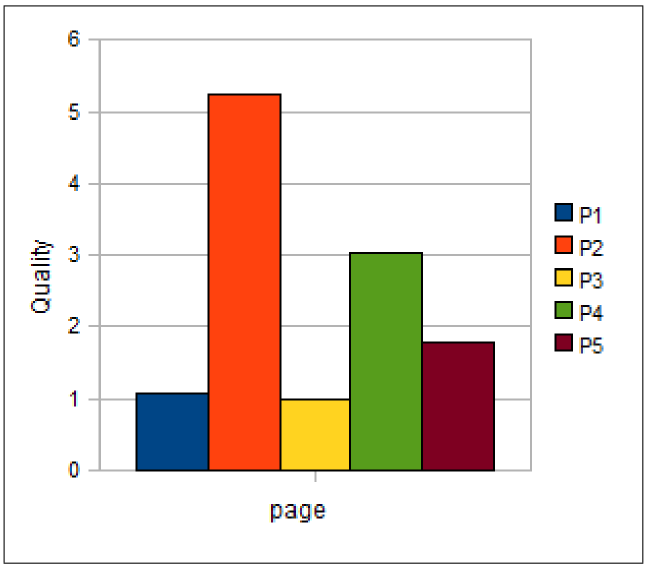


Figure 26: Average optimization results for all test pages

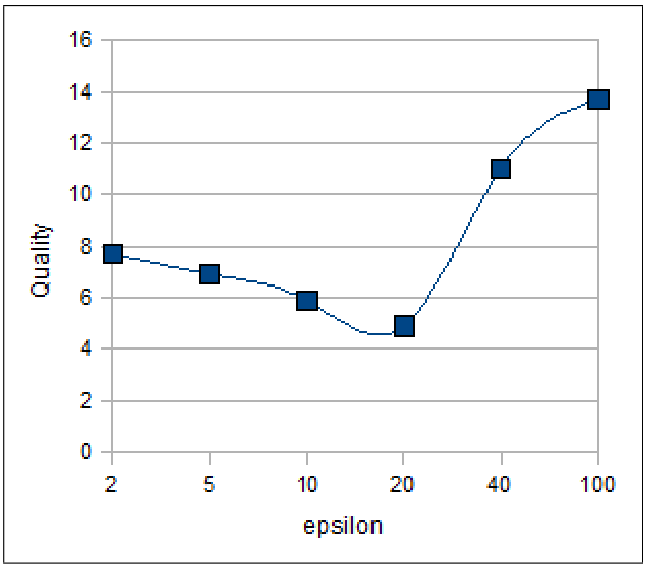


Figure 27: Results with different $\epsilon$ values for P4

width. As can been seen from both graphs the quality decreases but then at some point is increases - in case of the P4 instance even steeply. Even though the quality with an $\epsilon$ value of 20 yields better quality results for the P4 instance, to be sure not to conflict with the incline, a lower $\epsilon$ value of 10 was chosen for the main evaluation process. Even the result qualities for values lower than 10 are still better than result qualities for values above 40.

## 8.4 Investigation of different penalizations

Several methods were introduced to vary pixel penalization. For example with *heavy distance penalization* (HDP) one can square the distance to penalize pixels further apart more. Or *exact match favoritism* (EMF) rewards pixel which have a direct adjacent partner. Both methods were used in the main optimization process. To show the behavior without these modifications tests were made with all test instances. Figure 29 shows the normal test results that were made compared to runs made without HDP and EMF. It can be seen that enabling EBP and HDP for the most part improves the results.

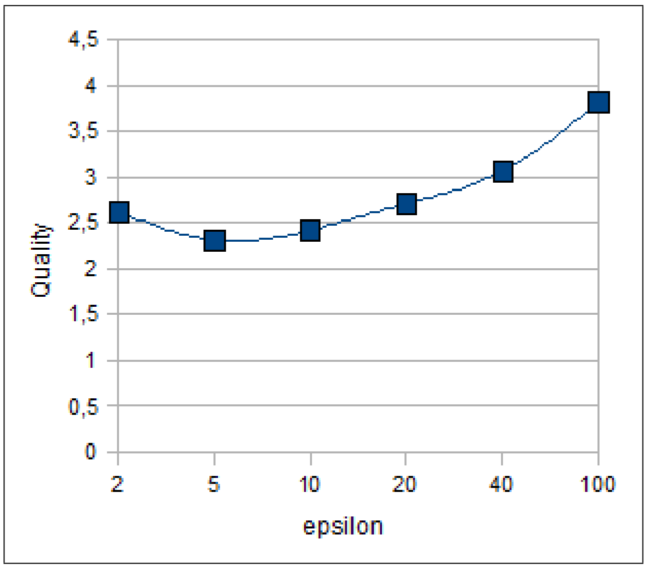


Figure 28: Average results with different $\epsilon$ values

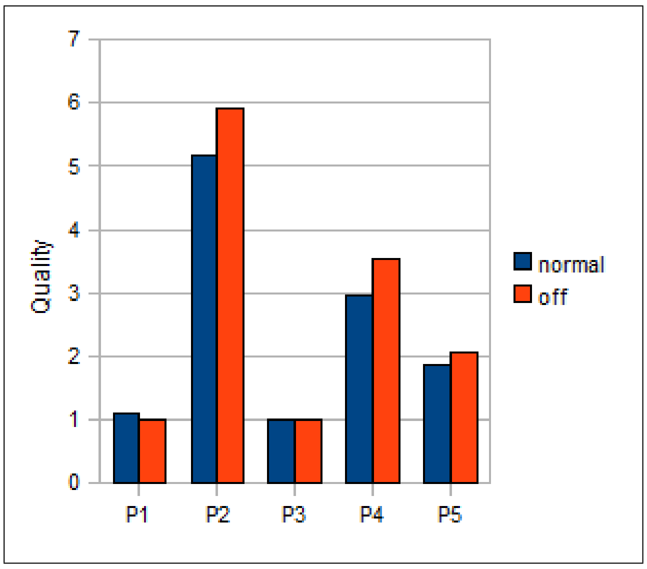


Figure 29: Results with different pixel penalization

## 8.5 Comparison of other optimization methods

It has already been mentioned that the VND optimization has been used mainly. To show off the results of the other implemented methods (iterated local search and simulated annealing) all test instances have been solved using all available optimization techniques. Figure 30 and 31 show graphs of the results. The vertical axis depict the quality of the solution. On the horizontal axis are triples of the solution methods.

Figure 30 shows specific results for test pages with 300 dpi and 20 pixel strip width. P1 and P3 could be solved by every method to optimality. It can be seen that VND here offers the best results, only in one instance P4 ILS gives better results.

Comparing this specific result with the average results of all test instances (Figure 31) shows that this claim can be supported. The instances P1 and P3 could be solved by VND better, with P2 VND leads only marginally. Also the average results show that for P4 ILS yields the best results, but only slightly. This should justify the motivation to further use VND as main optimization method.

## 8.6 Reconstruction of multiple pages

So far only the reconstruction of a single page has been looked upon. But of course also the recovering of multiple pages is of interest. First it is important to note that this is not a special case for either the evaluation or the reconstruction process. If shreds come from multiple pages the output can be interpreted as one wide page. Of course the natural order in which they were shredded cannot be restored. If for example three pages (A, B, C) were shredded every permutation of these pages would be an optimal solution. Optionally for reconstructing multiple pages *empty border penalization* (EBP) can be turned off, since it is imminent that empty borders must match somewhere.

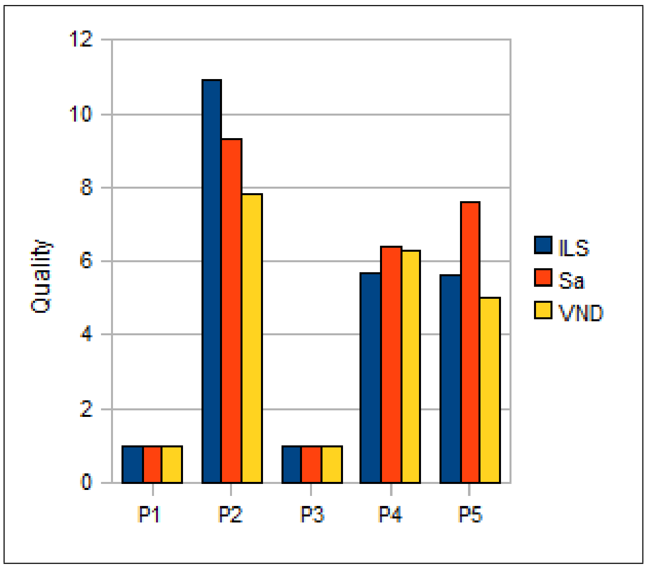


Figure 30: Results for ILS, SA and VND for 300dpi and 20 SW

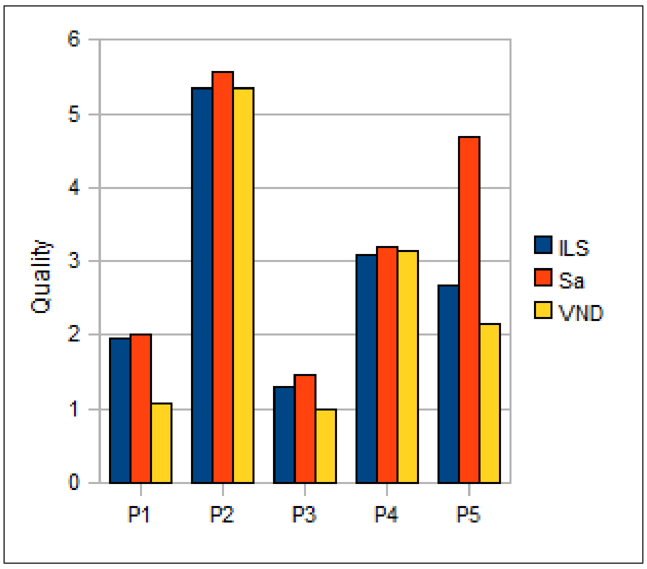


Figure 31: Average optimization results for ILS, SA and VND

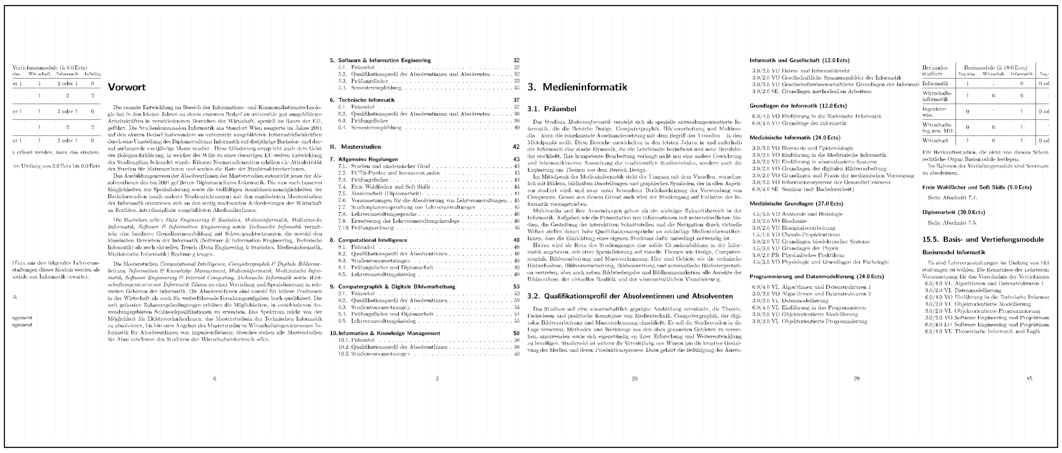


Figure 32: Result for a multiple page instance

To test this, all 600 dpi instances with 100 pixel strip width were put together and reconstructed. At first as test setup the same parameters as in the beginning of chapter 8 was chosen. Thereafter EBP was disabled. With these configurations solutions of the same quality could be found. A sample solution is presented in Figure 32. This problem has 168 strips and a solution quality of two was reached. Also other multiple page test instances were tried with and without EBP but from the results no clear decision can be made if deactivating EBP yields any significant improvements.

As can be seen from the shown solution larger parts of it were reconstructed just fine. The pages with much text on it are presented optimally. Also the test instance with the index table (which can be seen in Figure 17 more closely) was reconstructed correctly. The test instance with the table in it was cut in half and put on the left and right end of the solution. The cut through this page must somehow yield improvements at other places. It is also important to note that this found solution already has a better fitness value than the optimal solution.

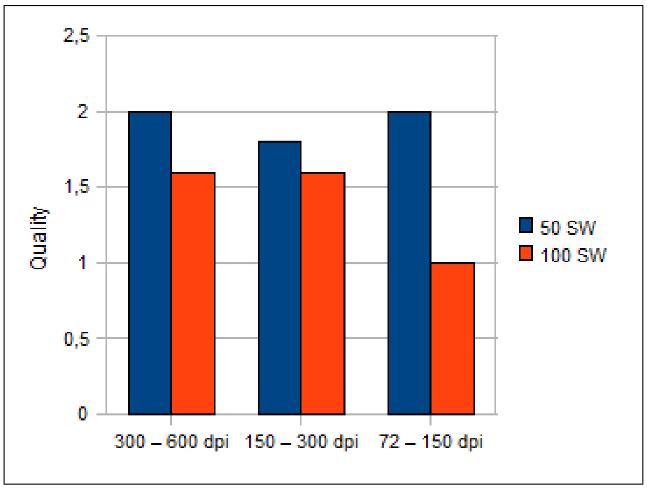


Figure 33: Comparing resolution with strip widths

## 8.7 Resolution and number of strips

Finally I want to discuss the topic of comparing results from different resolution instances. One cannot directly compare test instances having the same resolution or the same strip width since these problems may not be equally big. The 600 dpi test instance with 20 pixel strip width contains a lot more strips than the 72 dpi 20 pixel strip width instance. But it is perfectly legal to compare e.g. the two P1 instances having for one 150 dpi 50 pixel strip width and 300 dpi 100 pixel strip width. As can also be seen from the data in appendix A these two instances have the same number of strips. That makes a comparison reasonable. To accommodate this Figure 33 shows the summarized results of all comparable test instances. The horizontal axis displays groups of comparable instances. All 50 and 100 pixel strip width instances have been averaged and placed next to each other, since the instances in each group have the same number of strips. It can be seen that the higher resolution instances have been solved better then their lower resolution counterparts. This also supports the claim that higher resolution problems tend to be easier to solve.

# 9 Conclusion and future work

As can be seen from the concrete implementation the evaluation method presented offers a good approach to the reconstruction of strip shredded text documents (RSSTD). This method can theoretically be applied to other applications, not only to strip-shredded papers. If you can define a natural order between the elements you can even apply this to e.g. manually torn paper. This method may even be applied to normal images. Test instance Q1, which is a normal photograph, has been split into 16 strips and was solved correctly. So although I have not examined this type of application, there is at least potential here.

An important parameter for the algorithm is the variable $\epsilon$. This variable controls the coverage of the strip-neighborhood. For badly torn paper a high value is beneficial, for precisely cut paper a small value should be sufficient. Also the kind of writing on the paper may change the usage of $\epsilon$.

It has been observed that there are often long passages of correctly sequenced strips. If these could be bound together (merge into one big strip) the complexity of the problem could be broken down severely. This process can be done several times (merge already merged chumps). Of course it is not trivial to do that. One method to do this is to let a human operator control the merging process through a procedure called human in the loop. Another way would be to pass specific strips through OCR. This would allow the automatic verification if text on certain strips make out valid text (of course this would not work for images).

Since for small instances the method shows good results (i.e. no test instance with 100 pixel strip width has a quality above 5) it would be conceivable to implement a system that squeezes the problem size. Besides OCR or human operators, special heuristics could try to find groups or clusters of strips. By merging, new solutions could be found and evaluated. If no satisfying results are achieved with this clustering, this aggregation may be canceled and other clusters can be tried.

The biggest problem for the evaluation are unfavorable cuts at the edge of characters – e.g. on the far left of a capital D. The horizontal pixels at the border get drastically penalized. This makes it very hard to find the correct adjacent strip.

The reconstruction is by no mean a straightforward process. Often enough it happens that two strips that don't belong together have very good fitness. Tweaking is definitely necessary.

# A Test instances

Five representative pages (see appendix D) were extracted from a PDF document in different resolutions (72, 150, 300 and 600 pixels per inch). Each instance was strip-shredded with different strip sizes (100, 50 and 20 pixel strip width). The exact data about these instances are detailed in the following table.

| Instance | resolution[3] | strip width[4] | nr. of strips[5] | optimal obj. value[6] | Nr. |
|---|---|---|---|---|---|
| P1 | 72 | 100 | 6 (6) | -1149 | 1 |
| P1 | 72 | 50 | 10 (12) | -5351 | 2 |
| P1 | 72 | 20 | 22 (30) | -13590 | 3 |
| P1 | 150 | 100 | 10 (13) | -9472 | 4 |
| P1 | 150 | 50 | 19 (25) | -19744 | 5 |
| P1 | 150 | 20 | 44 (62) | -44538 | 6 |
| P1 | 300 | 100 | 19 (25) | -55683 | 7 |
| P1 | 300 | 50 | 36 (50) | -79679 | 8 |
| P1 | 300 | 20 | 88 (124) | -281686 | 9 |
| P1 | 600 | 100 | 36 (50) | -417669 | 10 |
| P1 | 600 | 50 | 71 (100) | -829171 | 11 |
| P1 | 600 | 20 | 174 (248) | -2088595 | 12 |
| P2 | 72 | 100 | 6 (6) | 12615 | 13 |
| P2 | 72 | 50 | 10 (12) | 16352 | 14 |
| P2 | 72 | 20 | 20 (30) | 27485 | 15 |
| P2 | 150 | 100 | 9 (13) | -13213 | 16 |
| P2 | 150 | 50 | 18 (25) | -1573 | 17 |
| P2 | 150 | 20 | 40 (62) | -2540 | 18 |
| P2 | 300 | 100 | 18 (25) | -44936 | 19 |

[3]in pixels per inch (ppi)
[4]in pixels
[5]in brackets is the number of strips without blank strip elimination, see chapter 4.5.1
[6]this is the objective value, if all strips have been ordered correctly

| | | | | | |
|---|---|---|---|---|---|
| P2 | 300 | 50 | 32 (50) | -94374 | 20 |
| P2 | 300 | 20 | 81 (124) | -185139 | 21 |
| P2 | 600 | 100 | 32 (50) | -255615 | 22 |
| P2 | 600 | 50 | 64 (100) | -487486 | 23 |
| P2 | 600 | 20 | 156 (248) | -1233012 | 24 |
| P3 | 72 | 100 | 6 (6) | -3847 | 25 |
| P3 | 72 | 50 | 10 (12) | -5425 | 26 |
| P3 | 72 | 20 | 22 (30) | -12169 | 27 |
| P3 | 150 | 100 | 10 (13) | -4281 | 28 |
| P3 | 150 | 50 | 19 (25) | -18416 | 29 |
| P3 | 150 | 20 | 44 (62) | -44290 | 30 |
| P3 | 300 | 100 | 19 (25) | -67129 | 31 |
| P3 | 300 | 50 | 36 (50) | -106179 | 32 |
| P3 | 300 | 20 | 88 (124) | -303138 | 33 |
| P3 | 600 | 100 | 36 (50) | -418178 | 34 |
| P3 | 600 | 50 | 71 (100) | -860327 | 35 |
| P3 | 600 | 20 | 174 (248) | -2163140 | 36 |
| P4 | 72 | 100 | 5 (6) | 13290 | 37 |
| P4 | 72 | 50 | 8 (12) | 14866 | 38 |
| P4 | 72 | 20 | 18 (30) | 16202 | 39 |
| P4 | 150 | 100 | 8 (13) | -1347 | 40 |
| P4 | 150 | 50 | 15 (25) | -3570 | 41 |
| P4 | 150 | 20 | 36 (62) | -22950 | 42 |
| P4 | 300 | 100 | 15 (25) | -42138 | 43 |
| P4 | 300 | 50 | 28 (50) | -78542 | 44 |
| P4 | 300 | 20 | 71 (124) | -211966 | 45 |
| P4 | 600 | 100 | 28 (50) | -246686 | 46 |
| P4 | 600 | 50 | 55 (100) | -531599 | 47 |
| P4 | 600 | 20 | 141 (248) | -1397631 | 48 |
| P5 | 72 | 100 | 6 (6) | 4642 | 49 |

| | | | | | |
|---|---:|---:|---:|---:|---:|
| P5 | 72 | 50 | 10 (12) | 1975 | 50 |
| P5 | 72 | 20 | 22 (30) | -843 | 51 |
| P5 | 150 | 100 | 10 (13) | -7731 | 52 |
| P5 | 150 | 50 | 19 (25) | -15488 | 53 |
| P5 | 150 | 20 | 44 (62) | -16207 | 54 |
| P5 | 300 | 100 | 19 (25) | -53601 | 55 |
| P5 | 300 | 50 | 36 (50) | -117523 | 56 |
| P5 | 300 | 20 | 88 (124) | -181757 | 57 |
| P5 | 600 | 100 | 36 (50) | -296860 | 58 |
| P5 | 600 | 50 | 71 (100) | -565107 | 59 |
| P5 | 600 | 20 | 175 (248) | -1392428 | 60 |

## B Construction heuristic data

The construction heuristics forward page construction FPC, duplex page construction DPC and randomized duplex page construction RDPC were tested. FPC and DPC were called for all possible situations (i.e. each strip was one time selected being first). RDPC was called 100 times the number of strips that existed for the specific instance.

The values in the $\alpha$ row mean the average fitness, the $\beta$ row represents average quality, the $\gamma$ row means minimum fitness and the $\delta$ row shows minimum quality. The values in brackets show the standard deviation.

| Inst. | FPC | DPC | RDPC | |
|---|---:|---:|---:|---|
| 1 | 1892.5(1463.28) | -1149(0) | 9651.36(4620.46) | $\alpha$ |
| | 1.83(0.37) | 1(0) | 2.9(0.71) | $\beta$ |
| | -1149 | -1149 | -1149 | $\gamma$ |
| | 1 | 1 | 1 | $\delta$ |

| | | | | |
|---|---|---|---|---|
| 2 | -2355.3(1164.06)<br>1.9(0.3)<br>-5351<br>1 | -1428.5(3922.5)<br>1.5(0.5)<br>-5351<br>1 | 14487.44(5218.45)<br>4.68(0.96)<br>-782<br>2 | $\alpha$<br>$\beta$<br>$\gamma$<br>$\delta$ |
| 3 | -5563.23(2085.24)<br>5.82(1.47)<br>-8653<br>3 | -4779.18(4537.54)<br>5.59(2.06)<br>-12288<br>2 | 21892.28(6763.7)<br>10.8(1.42)<br>3282<br>6 | $\alpha$<br>$\beta$<br>$\gamma$<br>$\delta$ |
| 4 | -2744.1(2473.13)<br>1.9(0.3)<br>-9472<br>1 | -3663.6(7113.81)<br>1.4(0.49)<br>-9472<br>1 | 38803.71(10376.92)<br>4.75(0.99)<br>6170<br>2 | $\alpha$<br>$\beta$<br>$\gamma$<br>$\delta$ |
| 5 | -13056.63(1841.88)<br>1.95(0.22)<br>-19744<br>1 | -13629.89(7169.42)<br>1.42(0.49)<br>-19744<br>1 | 85946.04(13299.74)<br>9.21(1.32)<br>27125<br>5 | $\alpha$<br>$\beta$<br>$\gamma$<br>$\delta$ |
| 6 | -25049.93(7820.02)<br>4.36(1.75)<br>-37198<br>2 | -8018.34(4081.63)<br>7.05(1.24)<br>-22036<br>4 | 212017.7(19051.72)<br>21.73(1.99)<br>143163<br>15 | $\alpha$<br>$\beta$<br>$\gamma$<br>$\delta$ |
| 7 | -22962.32(30654.58)<br>2.84(1.72)<br>-55683<br>1 | -55683(0)<br>1(0)<br>-55683<br>1 | 251179.88(31623.23)<br>9.27(1.34)<br>109388<br>5 | $\alpha$<br>$\beta$<br>$\gamma$<br>$\delta$ |
| 8 | -18016.17(41496.99)<br>4.67(2.26)<br>-79679<br>1 | -79679(0)<br>1(0)<br>-79679<br>1 | 518857.21(33298.26)<br>17.85(1.83)<br>377769<br>11 | $\alpha$<br>$\beta$<br>$\gamma$<br>$\delta$ |
| 9 | -220293.03(34313.25)<br>4.69(2.05)<br>-281686<br>1 | -281686(0)<br>1(0)<br>-281686<br>1 | 1367927.24(43519.49)<br>43.89(2.84)<br>1182863<br>32 | $\alpha$<br>$\beta$<br>$\gamma$<br>$\delta$ |
| 10 | -304481.42(34478.04)<br>3.64(0.71)<br>-417669<br>1 | -417669(0)<br>1(0)<br>-417669<br>1 | 1333517.57(96054.81)<br>17.57(1.84)<br>892171<br>11 | $\alpha$<br>$\beta$<br>$\gamma$<br>$\delta$ |
| 11 | -679950.66(78753.39)<br>4.48(1.85)<br>-829171<br>1 | -829171(0)<br>1(0)<br>-829171<br>1 | 2852535.05(111077.63)<br>35.31(2.56)<br>2386133<br>27 | $\alpha$<br>$\beta$<br>$\gamma$<br>$\delta$ |
| 12 | -1864185.23(108532.16)<br>6.27(2.39)<br>-2088595<br>1 | -2088595(0)<br>1(0)<br>-2088595<br>1 | 7472977.43(150414.97)<br>86.57(3.91)<br>6839763<br>71 | $\alpha$<br>$\beta$<br>$\gamma$<br>$\delta$ |

| | | | | |
|---|---|---|---|---|
| 13 | 18372.17(5105.84) | 2567(2795.24) | 12530.68(6792.45) | $\alpha$ |
| | 2.5(0.76) | 2.33(0.75) | 3.12(0.79) | $\beta$ |
| | 12144 | 628 | 628 | $\gamma$ |
| | 1 | 2 | 2 | $\delta$ |
| 14 | 18851(7641.68) | 9287.7(4249.18) | 25062.25(8008.91) | $\alpha$ |
| | 3.3(0.9) | 3.5(1.02) | 5.3(0.97) | $\beta$ |
| | 4386 | 4365 | 5166 | $\gamma$ |
| | 1 | 2 | 2 | $\delta$ |
| 15 | 19397.35(3969.84) | 15430.85(4368.88) | 44901.66(8612.98) | $\alpha$ |
| | 7.4(1.07) | 8.35(1.39) | 9.89(1.39) | $\beta$ |
| | 14848 | 10001 | 18552 | $\gamma$ |
| | 6 | 5 | 5 | $\delta$ |
| 16 | -862.56(12309.16) | -13213(0) | 33555.95(20303.36) | $\alpha$ |
| | 2.67(0.67) | 1(0) | 4.06(0.92) | $\beta$ |
| | -13213 | -13213 | -12074 | $\gamma$ |
| | 1 | 1 | 2 | $\delta$ |
| 17 | 17810.61(20584.27) | -17435.78(709.14) | 108042.16(22396.74) | $\alpha$ |
| | 4.22(1.65) | 3.78(0.42) | 8.94(1.39) | $\beta$ |
| | -11771 | -18570 | 27757 | $\gamma$ |
| | 1 | 3 | 5 | $\delta$ |
| 18 | 28176.45(9695.2) | 12062.85(6323.05) | 240676.95(30005.23) | $\alpha$ |
| | 12.25(1.26) | 10.35(1.84) | 19.75(1.96) | $\beta$ |
| | 12556 | 2685 | 143093 | $\gamma$ |
| | 10 | 7 | 13 | $\delta$ |
| 19 | -30945.78(28223.21) | -48672.33(6824.04) | 218387.68(52721.08) | $\alpha$ |
| | 3.06(0.7) | 2.11(0.31) | 8.81(1.38) | $\beta$ |
| | -62272 | -64693 | 43109 | $\gamma$ |
| | 1 | 2 | 4 | $\delta$ |
| 20 | -80694.5(25841.05) | -97938.28(5992.5) | 456289.98(60559.13) | $\alpha$ |
| | 3.06(0.56) | 1.62(1.08) | 15.87(1.82) | $\beta$ |
| | -112369 | -113464 | 242984 | $\gamma$ |
| | 1 | 1 | 10 | $\delta$ |
| 21 | -154090.2(38288.46) | -225070.04(6173.75) | 1218772.25(93482.02) | $\alpha$ |
| | 13.17(3.71) | 9.21(1.45) | 40.55(2.78) | $\beta$ |
| | -222510 | -230814 | 909971 | $\gamma$ |
| | 8 | 8 | 29 | $\delta$ |
| 22 | -232557.16(48518) | -278677.44(17927.44) | 1091843.22(148349.86) | $\alpha$ |
| | 3.09(0.58) | 2.88(1.45) | 15.67(1.75) | $\beta$ |
| | -295777 | -298229 | 510583 | $\gamma$ |
| | 1 | 1 | 10 | $\delta$ |
| 23 | -536288.14(62928.87) | -591336.2(17412.35) | 2220875.55(168988.04) | $\alpha$ |
| | 6.86(0.88) | 6.81(1.38) | 31.78(2.37) | $\beta$ |
| | -611550 | -609883 | 1501967 | $\gamma$ |
| | 5 | 4 | 23 | $\delta$ |

| | | | | |
|---|---|---|---|---|
| 24 | -1394874.83(62199.99) | -1399266.35(58136.51) | 5845497.9(253658.41) | $\alpha$ |
| | 16.45(1.43) | 14.65(1.41) | 77.9(3.72) | $\beta$ |
| | -1484189 | -1489251 | 4754484 | $\gamma$ |
| | 13 | 12 | 62 | $\delta$ |
| 25 | 3453.5(3064.34) | 3763(5381.08) | 8634.84(5631.6) | $\alpha$ |
| | 2.67(0.75) | 2.33(0.94) | 2.82(0.77) | $\beta$ |
| | 384 | -3847 | -3847 | $\gamma$ |
| | 2 | 1 | 1 | $\delta$ |
| 26 | 1639.8(3237.44) | 2096.2(4407.8) | 13385.45(5337.54) | $\alpha$ |
| | 3.2(0.98) | 2.9(0.7) | 4.86(0.96) | $\beta$ |
| | -2869 | -5425 | -5425 | $\gamma$ |
| | 2 | 1 | 1 | $\delta$ |
| 27 | -617(3041.16) | -3973.91(1887.26) | 31338.95(7593.7) | $\alpha$ |
| | 7.32(1.39) | 4.55(1.53) | 10.7(1.46) | $\beta$ |
| | -8725 | -8704 | 5663 | $\gamma$ |
| | 5 | 3 | 6 | $\delta$ |
| 28 | 3044.7(3053.98) | -4281(0) | 48979.36(12714.44) | $\alpha$ |
| | 1.9(0.3) | 1(0) | 4.76(1.01) | $\beta$ |
| | -4281 | -4281 | -4281 | $\gamma$ |
| | 1 | 1 | 1 | $\delta$ |
| 29 | -10857.79(2358.37) | -18416(0) | 99542.56(15214.51) | $\alpha$ |
| | 1.95(0.22) | 1(0) | 9.3(1.36) | $\beta$ |
| | -18416 | -18416 | 47856 | $\gamma$ |
| | 1 | 1 | 5 | $\delta$ |
| 30 | -4237.32(13582.67) | -28523.93(20019.49) | 257826.59(22519.84) | $\alpha$ |
| | 6.34(1.76) | 3.09(2.91) | 21.69(2.05) | $\beta$ |
| | -28849 | -44290 | 186132 | $\gamma$ |
| | 2 | 1 | 15 | $\delta$ |
| 31 | -37474.47(17270.04) | -67129(0) | 263333.17(32633.51) | $\alpha$ |
| | 2.58(1.14) | 1(0) | 9.31(1.38) | $\beta$ |
| | -67129 | -67129 | 120184 | $\gamma$ |
| | 1 | 1 | 5 | $\delta$ |
| 32 | -44828.31(30553.97) | -106179(0) | 538114.94(36462.74) | $\alpha$ |
| | 4.81(2.04) | 1(0) | 17.82(1.84) | $\beta$ |
| | -106179 | -106179 | 370782 | $\gamma$ |
| | 1 | 1 | 11 | $\delta$ |
| 33 | -195443.84(34680.42) | -303138(0) | 1482133.29(58688.51) | $\alpha$ |
| | 6.92(2.21) | 1(0) | 44.11(2.88) | $\beta$ |
| | -303138 | -303138 | 1265515 | $\gamma$ |
| | 1 | 1 | 33 | $\delta$ |
| 34 | -280368.58(59231.82) | -418178(0) | 1361053.78(94483.14) | $\alpha$ |
| | 4.03(1.3) | 1(0) | 17.98(1.85) | $\beta$ |
| | -418178 | -418178 | 958216 | $\gamma$ |
| | 1 | 1 | 10 | $\delta$ |

| | | | | |
|---|---|---|---|---|
| 35 | -750732.07(29650.51) | -860327(0) | 2950970.58(112015.25) | $\alpha$ |
| | 3.59(0.6) | 1(0) | 35.31(2.58) | $\beta$ |
| | -860327 | -860327 | 2468814 | $\gamma$ |
| | 1 | 1 | 25 | $\delta$ |
| 36 | -1912991.99(114422.31) | -2163140(0) | 7715833.66(198223.27) | $\alpha$ |
| | 6.71(2.63) | 1(0) | 86.91(3.96) | $\beta$ |
| | -2163140 | -2163140 | 6958067 | $\gamma$ |
| | 1 | 1 | 69 | $\delta$ |
| 37 | 4993.6(3066.2) | 3821.4(1269.2) | 9432.94(4744.98) | $\alpha$ |
| | 2.6(0.49) | 2.2(0.4) | 2.89(0.62) | $\beta$ |
| | 1283 | 1283 | 1283 | $\gamma$ |
| | 2 | 2 | 2 | $\delta$ |
| 38 | 11934.25(5649.53) | 3015.62(852.89) | 18427.07(6762.91) | $\alpha$ |
| | 3.12(0.6) | 2.62(0.48) | 4.06(0.81) | $\beta$ |
| | 2958 | 2336 | 2336 | $\gamma$ |
| | 2 | 2 | 2 | $\delta$ |
| 39 | 17313.11(4574.44) | 5944.33(2012.17) | 45130.6(7461.71) | $\alpha$ |
| | 4.89(0.94) | 4.56(1.46) | 8.97(1.31) | $\beta$ |
| | 5995 | 3110 | 22881 | $\gamma$ |
| | 3 | 3 | 5 | $\delta$ |
| 40 | 6398.62(3256.77) | -1347(0) | 32947.09(10871.91) | $\alpha$ |
| | 2.12(0.6) | 1(0) | 3.65(0.93) | $\beta$ |
| | -1347 | -1347 | -1347 | $\gamma$ |
| | 1 | 1 | 1 | $\delta$ |
| 41 | 1374.67(9734.03) | -4211.93(909.53) | 74184.34(13291.15) | $\alpha$ |
| | 3.33(1.01) | 2.6(0.71) | 7.31(1.26) | $\beta$ |
| | -4711 | -4711 | 24546 | $\gamma$ |
| | 3 | 1 | 4 | $\delta$ |
| 42 | -28942.22(1559.85) | -22340.36(7883.42) | 173476.67(20843.52) | $\alpha$ |
| | 3.19(0.62) | 3.58(0.76) | 18.2(1.9) | $\beta$ |
| | -31037 | -31206 | 110048 | $\gamma$ |
| | 2 | 3 | 12 | $\delta$ |
| 43 | -37729.13(4328.45) | -40155.47(3966.73) | 151898.41(40100.47) | $\alpha$ |
| | 2.93(0.25) | 2.73(0.68) | 7.21(1.25) | $\beta$ |
| | -42139 | -42139 | -22060 | $\gamma$ |
| | 2 | 1 | 4 | $\delta$ |
| 44 | -73918.96(4021.09) | -76755.61(3790.74) | 355541.7(44328.19) | $\alpha$ |
| | 3.11(0.56) | 2.89(0.31) | 13.93(1.68) | $\beta$ |
| | -78543 | -78543 | 199087 | $\gamma$ |
| | 2 | 2 | 8 | $\delta$ |
| 45 | -181632.38(20663.78) | -169178.65(12360.71) | 969219.59(62901.58) | $\alpha$ |
| | 6.76(0.68) | 6.14(0.48) | 35.34(2.6) | $\beta$ |
| | -209769 | -205930 | 734928 | $\gamma$ |
| | 5 | 5 | 26 | $\delta$ |

| | | | | |
|---|---:|---:|---:|---|
| 46 | -196336.96(35608.59) | -245475.71(2964.58) | 858640.14(123390.83) | $\alpha$ |
| | 2.96(0.63) | 1.14(0.35) | 13.68(1.67) | $\beta$ |
| | -246686 | -246686 | 427957 | $\gamma$ |
| | 1 | 1 | 8 | $\delta$ |
| 47 | -519902.09(9943.81) | -533995(0) | 1938669.7(164974.97) | $\alpha$ |
| | 2.24(0.69) | 2(0) | 27.24(2.29) | $\beta$ |
| | -533995 | -533995 | 1274533 | $\gamma$ |
| | 1 | 2 | 17 | $\delta$ |
| 48 | -1352938.64(55315.33) | -1385955.35(15809.89) | 5314013.04(255548.87) | $\alpha$ |
| | 6.87(0.99) | 6.32(0.9) | 70.55(3.58) | $\beta$ |
| | -1434013 | -1415043 | 4388679 | $\gamma$ |
| | 4 | 6 | 57 | $\delta$ |
| 49 | 9371.83(2955.28) | 3956.5(2387) | 12534.24(3980.87) | $\alpha$ |
| | 2.67(0.75) | 2(0) | 3.07(0.79) | $\beta$ |
| | 2889 | 2889 | 2889 | $\gamma$ |
| | 2 | 2 | 2 | $\delta$ |
| 50 | 14921.5(9690.65) | 2755.2(3423.32) | 22331.46(5912.38) | $\alpha$ |
| | 3.9(1.97) | 2.1(0.3) | 4.69(0.99) | $\beta$ |
| | -142 | -142 | 3258 | $\gamma$ |
| | 2 | 2 | 2 | $\delta$ |
| 51 | 13527.68(5123.01) | 301.45(2677.38) | 45138.25(6949.45) | $\alpha$ |
| | 7.45(1.8) | 3.41(1.23) | 10.79(1.55) | $\beta$ |
| | 915 | -1546 | 21665 | $\gamma$ |
| | 5 | 2 | 6 | $\delta$ |
| 52 | 4187(6060.29) | -7187.5(1630.5) | 44619.14(13854.75) | $\alpha$ |
| | 2.6(1.11) | 1.1(0.3) | 4.67(1.04) | $\beta$ |
| | -7731 | -7731 | -3408 | $\gamma$ |
| | 1 | 1 | 2 | $\delta$ |
| 53 | -4329.11(8252.17) | -15027.05(1343.88) | 86722.27(17019.27) | $\alpha$ |
| | 3.05(1.57) | 1.21(0.61) | 9.5(1.42) | $\beta$ |
| | -15488 | -15488 | 34719 | $\gamma$ |
| | 1 | 1 | 5 | $\delta$ |
| 54 | 14886.52(10666.68) | -2399.5(19149.21) | 234825.66(27348.22) | $\alpha$ |
| | 13.73(3.08) | 10.05(1.52) | 21.93(2.04) | $\beta$ |
| | -35342 | -35342 | 144447 | $\gamma$ |
| | 8 | 8 | 14 | $\delta$ |
| 55 | -23553.84(21232.38) | -36021.84(14991.56) | 151014.59(30041.34) | $\alpha$ |
| | 3.89(1.77) | 4.47(2.96) | 9.21(1.41) | $\beta$ |
| | -53601 | -53601 | 48914 | $\gamma$ |
| | 1 | 1 | 5 | $\delta$ |
| 56 | -86256.97(10258.61) | -108967.83(7230.44) | 372520.31(45136.09) | $\alpha$ |
| | 7.44(2.22) | 3.33(1.97) | 17.81(1.92) | $\beta$ |
| | -102791 | -117523 | 217052 | $\gamma$ |
| | 3 | 1 | 12 | $\delta$ |

| | | | | |
|---|---|---|---|---|
| 57 | -38849.83(45485.76) | -126086.51(58371.04) | 1145547.3(83437.21) | $\alpha$ |
| | 10.34(2.18) | 10.33(3.6) | 43.86(2.78) | $\beta$ |
| | -145316 | -194305 | 835358 | $\gamma$ |
| | 7 | 5 | 33 | $\delta$ |
| 58 | -229664.69(31066.72) | -296860(0) | 877493.15(103749.97) | $\alpha$ |
| | 3.94(1.76) | 1(0) | 17.76(1.92) | $\beta$ |
| | -296860 | -296860 | 522550 | $\gamma$ |
| | 1 | 1 | 11 | $\delta$ |
| 59 | -500863.63(24998.26) | -565107(0) | 1936177.87(131244.41) | $\alpha$ |
| | 3.96(1.67) | 1(0) | 35.33(2.57) | $\beta$ |
| | -565107 | -565107 | 1431536 | $\gamma$ |
| | 1 | 1 | 26 | $\delta$ |
| 60 | -1214061.28(85111.9) | -1355338.98(12291.21) | 5161064.45(181366.38) | $\alpha$ |
| | 13.42(3.71) | 8.25(2.56) | 87.28(3.97) | $\beta$ |
| | -1362036 | -1390284 | 4391758 | $\gamma$ |
| | 4 | 2 | 72 | $\delta$ |

## C Test results

The tests have been made using multistart variable neighborhood descent. The sample for the average consisted of 30 consecutive runs. The instances with strip width of 100 were given 15 seconds, the ones with strip width of 50 were given 30 seconds and the ones with strip width of 20 were given 90 seconds to find a solution. Tests were made on an AMD Dual-Core Opteron 2214, 2,2 GHz, 4 GByte RAM and Java 1.5. Values in brackets show the standard deviation.

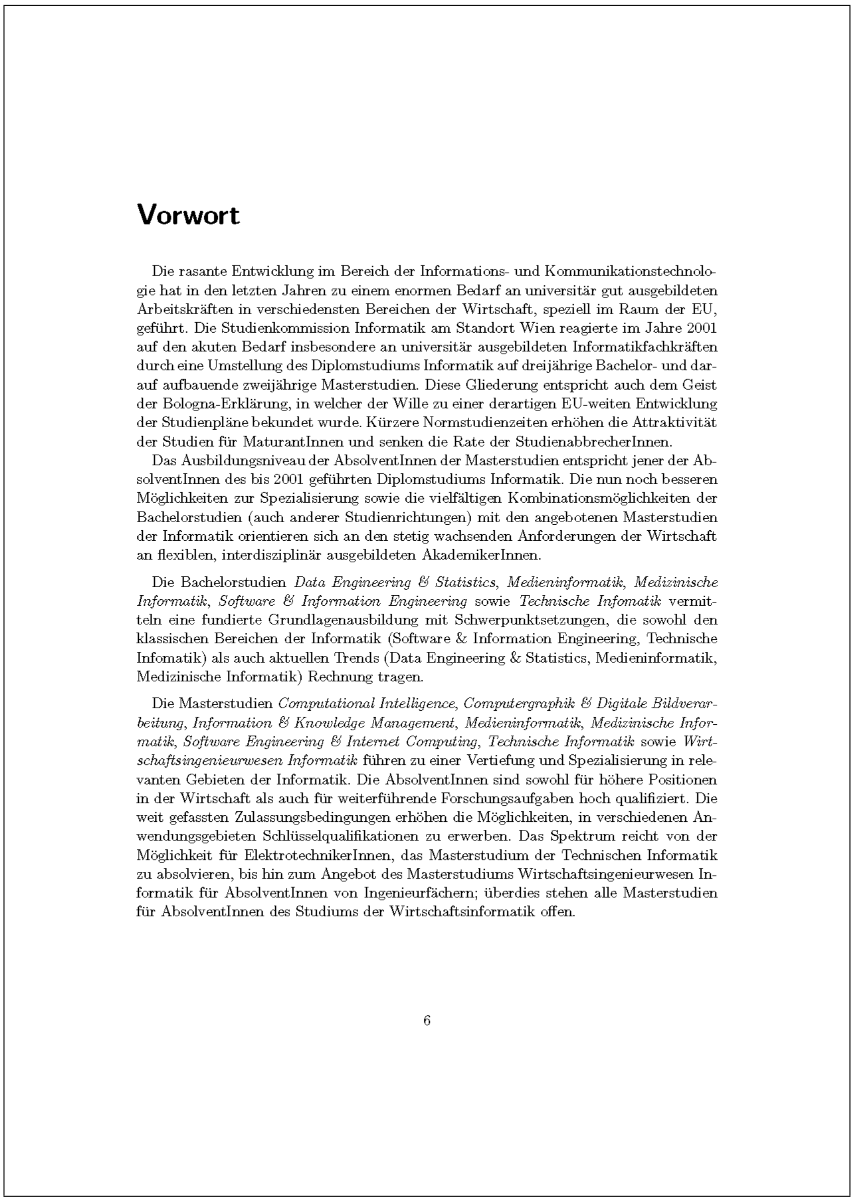
| Instance | avg. fitness | avg. quality | min. fitness | min. quality |
|---|---|---|---|---|
| 1 | -1149 (0) | 1 (0) | -1149 | 1 |
| 2 | -5351 (0) | 1 (0) | -5351 | 1 |
| 3 | -13765 (0) | 2 (0) | -13765 | 2 |
| 4 | -9472 (0) | 1 (0) | -9472 | 1 |
| 5 | -19744 (0) | 1 (0) | -19744 | 1 |

| | | | | |
|---|---:|---:|---:|---:|
| 6 | -44538 (0) | 1 (0) | -44538 | 1 |
| 7 | -55683 (0) | 1 (0) | -55683 | 1 |
| 8 | -79679 (0) | 1 (0) | -79679 | 1 |
| 9 | -281686 (0) | 1 (0) | -281686 | 1 |
| 10 | -417669 (0) | 1 (0) | -417669 | 1 |
| 11 | -829171 (0) | 1 (0) | -829171 | 1 |
| 12 | -2088595 (0) | 1 (0) | -2088595 | 1 |
| 13 | 628 (0) | 2 (0) | 628 | 2 |
| 14 | 3735 (0) | 4 (0) | 3735 | 4 |
| 15 | 3096 (0) | 6 (0) | 3096 | 6 |
| 16 | -13213 (0) | 1 (0) | -13213 | 1 |
| 17 | -18570 (0) | 3 (0) | -18570 | 3 |
| 18 | -31708.2 (9.6) | 6.8 (0.4) | -31713 | 6 |
| 19 | -64693 (0) | 2 (0) | -64693 | 2 |
| 20 | -114705.4 (364.8) | 4 (0) | -114827 | 4 |
| 21 | -254911.9 (534.11) | 9.7 (0.78) | -255608 | 9 |
| 22 | -298229 (0) | 4 (0) | -298229 | 4 |
| 23 | -614289.2 (357.61) | 5.9 (0.3) | -614707 | 5 |
| 24 | -1536855.2 (1824.06) | 14.5 (0.92) | -1539632 | 13 |
| 25 | -3847 (0) | 1 (0) | -3847 | 1 |
| 26 | -5425 (0) | 1 (0) | -5425 | 1 |
| 27 | -12169 (0) | 1 (0) | -12169 | 1 |
| 28 | -4281 (0) | 1 (0) | -4281 | 1 |
| 29 | -18416 (0) | 1 (0) | -18416 | 1 |
| 30 | -44290 (0) | 1 (0) | -44290 | 1 |
| 31 | -67129 (0) | 1 (0) | -67129 | 1 |
| 32 | -106179 (0) | 1 (0) | -106179 | 1 |
| 33 | -303138 (0) | 1 (0) | -303138 | 1 |
| 34 | -418178 (0) | 1 (0) | -418178 | 1 |
| 35 | -860327 (0) | 1 (0) | -860327 | 1 |

| | | | | |
|---|---|---|---|---|
| 36 | -2163140 (0) | 1 (0) | -2163140 | 1 |
| 37 | 1283 (0) | 3 (0) | 1283 | 3 |
| 38 | 2336 (0) | 2 (0) | 2336 | 2 |
| 39 | 1238 (0) | 4 (0) | 1238 | 4 |
| 40 | -1347 (0) | 1 (0) | -1347 | 1 |
| 41 | -4711 (0) | 3 (0) | -4711 | 3 |
| 42 | -31748 (0) | 4 (0) | -31748 | 4 |
| 43 | -42139 (0) | 3 (0) | -42139 | 3 |
| 44 | -78543 (0) | 3 (0) | -78543 | 3 |
| 45 | -217648.1 (820.91) | 5 (1.41) | -218027 | 3 |
| 46 | -246686 (0) | 1 (0) | -246686 | 1 |
| 47 | -533995 (0) | 2 (0) | -533995 | 2 |
| 48 | -1446094.7 (916.63) | 5.5 (0.81) | -1447652 | 4 |
| 49 | 2889 (0) | 2 (0) | 2889 | 2 |
| 50 | -142 (0) | 2 (0) | -142 | 2 |
| 51 | -3369 (0) | 2 (0) | -3369 | 2 |
| 52 | -7731 (0) | 1 (0) | -7731 | 1 |
| 53 | -15488 (0) | 1 (0) | -15488 | 1 |
| 54 | -40427 (0) | 4 (0) | -40427 | 4 |
| 55 | -53601 (0) | 1 (0) | -53601 | 1 |
| 56 | -117523 (0) | 1 (0) | -117523 | 1 |
| 57 | -236757.9 (3967.66) | 4.4 (1.11) | -243629 | 3 |
| 58 | -296860 (0) | 1 (0) | -296860 | 1 |
| 59 | -565107 (0) | 1 (0) | -565107 | 1 |
| 60 | -1392428 (0) | 1 (0) | -1392428 | 1 |

## D Test images

The following test pages were used to evaluate the reconstruction implementation. These pages reflect different types of page styles. Page P1 reflects a typical test page with a headline, P2 is a table of contents page, P3 is a text page with several headlines, P4 displays a listing and P5 displays a page with a table. Test instance Q1 is a photograph.



## Vorwort

Die rasante Entwicklung im Bereich der Informations- und Kommunikationstechnologie hat in den letzten Jahren zu einem enormen Bedarf an universitär gut ausgebildeten Arbeitskräften in verschiedensten Bereichen der Wirtschaft, speziell im Raum der EU, geführt. Die Studienkommission Informatik am Standort Wien reagierte im Jahre 2001 auf den akuten Bedarf insbesondere an universitär ausgebildeten Informatikfachkräften durch eine Umstellung des Diplomstudiums Informatik auf dreijährige Bachelor- und darauf aufbauende zweijährige Masterstudien. Diese Gliederung entspricht auch dem Geist der Bologna-Erklärung, in welcher der Wille zu einer derartigen EU-weiten Entwicklung der Studienpläne bekundet wurde. Kürzere Normstudienzeiten erhöhen die Attraktivität der Studien für MaturantInnen und senken die Rate der StudienabbrecherInnen.

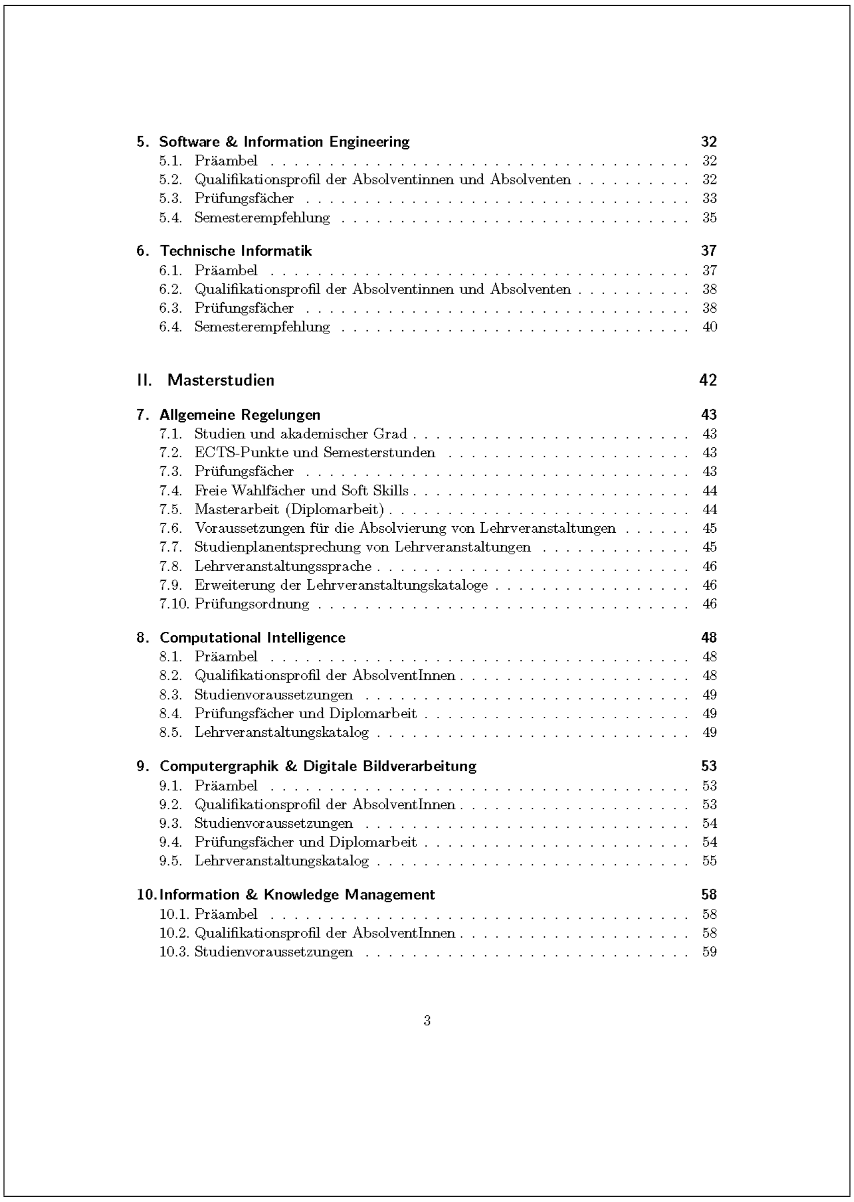
Das Ausbildungsniveau der AbsolventInnen der Masterstudien entspricht jener der AbsolventInnen des bis 2001 geführten Diplomstudiums Informatik. Die nun noch besseren Möglichkeiten zur Spezialisierung sowie die vielfältigen Kombinationsmöglichkeiten der Bachelorstudien (auch anderer Studienrichtungen) mit den angebotenen Masterstudien der Informatik orientieren sich an den stetig wachsenden Anforderungen der Wirtschaft an flexiblen, interdisziplinär ausgebildeten AkademikerInnen.

Die Bachelorstudien *Data Engineering & Statistics*, *Medieninformatik*, *Medizinische Informatik*, *Software & Information Engineering* sowie *Technische Infomatik* vermitteln eine fundierte Grundlagenausbildung mit Schwerpunktsetzungen, die sowohl den klassischen Bereichen der Informatik (Software & Information Engineering, Technische Infomatik) als auch aktuellen Trends (Data Engineering & Statistics, Medieninformatik, Medizinische Informatik) Rechnung tragen.

Die Masterstudien *Computational Intelligence*, *Computergraphik & Digitale Bildverarbeitung*, *Information & Knowledge Management*, *Medieninformatik*, *Medizinische Informatik*, *Software Engineering & Internet Computing*, *Technische Informatik* sowie *Wirtschaftsingenieurwesen Informatik* führen zu einer Vertiefung und Spezialisierung in relevanten Gebieten der Informatik. Die AbsolventInnen sind sowohl für höhere Positionen in der Wirtschaft als auch für weiterführende Forschungsaufgaben hoch qualifiziert. Die weit gefassten Zulassungsbedingungen erhöhen die Möglichkeiten, in verschiedenen Anwendungsgebieten Schlüsselqualifikationen zu erwerben. Das Spektrum reicht von der Möglichkeit für ElektrotechnikerInnen, das Masterstudium der Technischen Informatik zu absolvieren, bis hin zum Angebot des Masterstudiums Wirtschaftsingenieurwesen Informatik für AbsolventInnen von Ingenieurfächern; überdies stehen alle Masterstudien für AbsolventInnen des Studiums der Wirtschaftsinformatik offen.

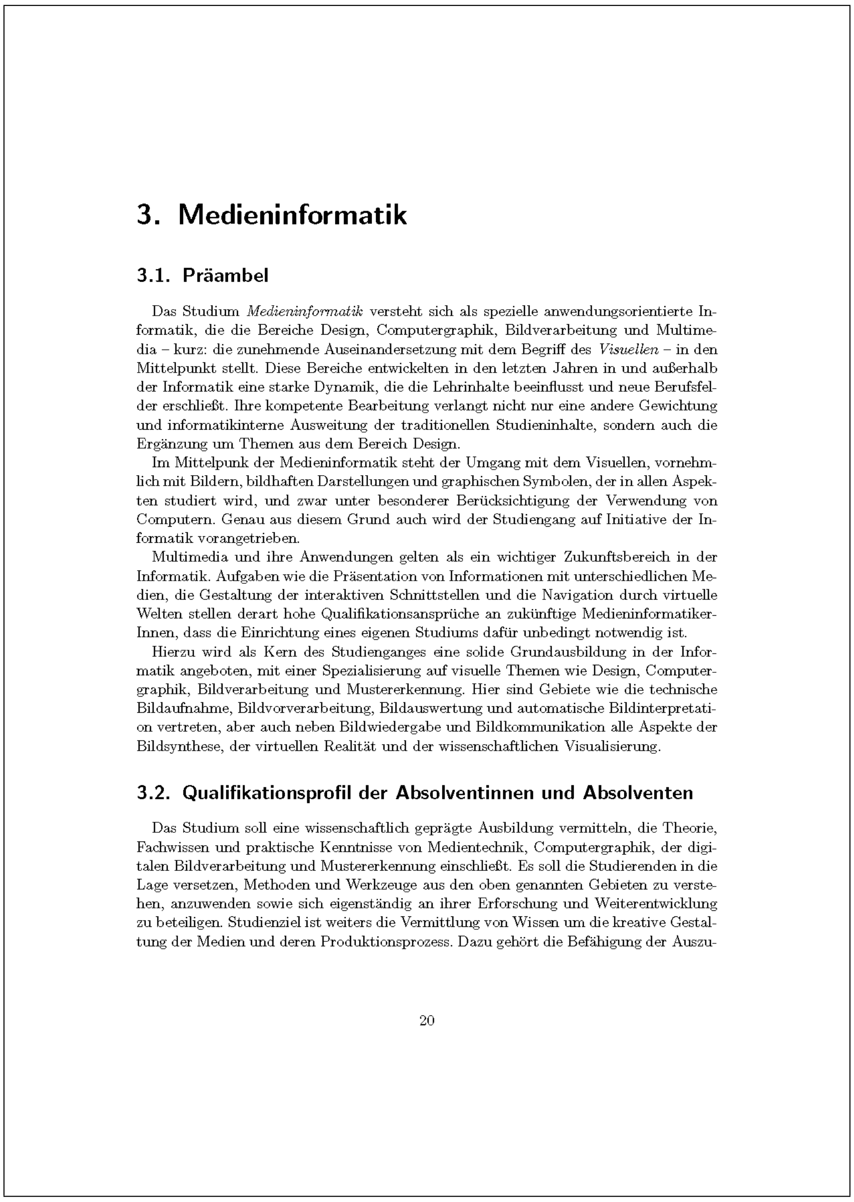
6

Figure 34: Test page P1



3

Figure 35: Test page P2



## 3. Medieninformatik

### 3.1. Präambel

Das Studium *Medieninformatik* versteht sich als spezielle anwendungsorientierte Informatik, die die Bereiche Design, Computergraphik, Bildverarbeitung und Multimedia – kurz: die zunehmende Auseinandersetzung mit dem Begriff des *Visuellen* – in den Mittelpunkt stellt. Diese Bereiche entwickelten in den letzten Jahren in und außerhalb der Informatik eine starke Dynamik, die die Lehrinhalte beeinflusst und neue Berufsfelder erschließt. Ihre kompetente Bearbeitung verlangt nicht nur eine andere Gewichtung und informatikinterne Ausweitung der traditionellen Studieninhalte, sondern auch die Ergänzung um Themen aus dem Bereich Design.

Im Mittelpunk der Medieninformatik steht der Umgang mit dem Visuellen, vornehmlich mit Bildern, bildhaften Darstellungen und graphischen Symbolen, der in allen Aspekten studiert wird, und zwar unter besonderer Berücksichtigung der Verwendung von Computern. Genau aus diesem Grund auch wird der Studiengang auf Initiative der Informatik vorangetrieben.

Multimedia und ihre Anwendungen gelten als ein wichtiger Zukunftsbereich in der Informatik. Aufgaben wie die Präsentation von Informationen mit unterschiedlichen Medien, die Gestaltung der interaktiven Schnittstellen und die Navigation durch virtuelle Welten stellen derart hohe Qualifikationsansprüche an zukünftige MedieninformatikerInnen, dass die Einrichtung eines eigenen Studiums dafür unbedingt notwendig ist.

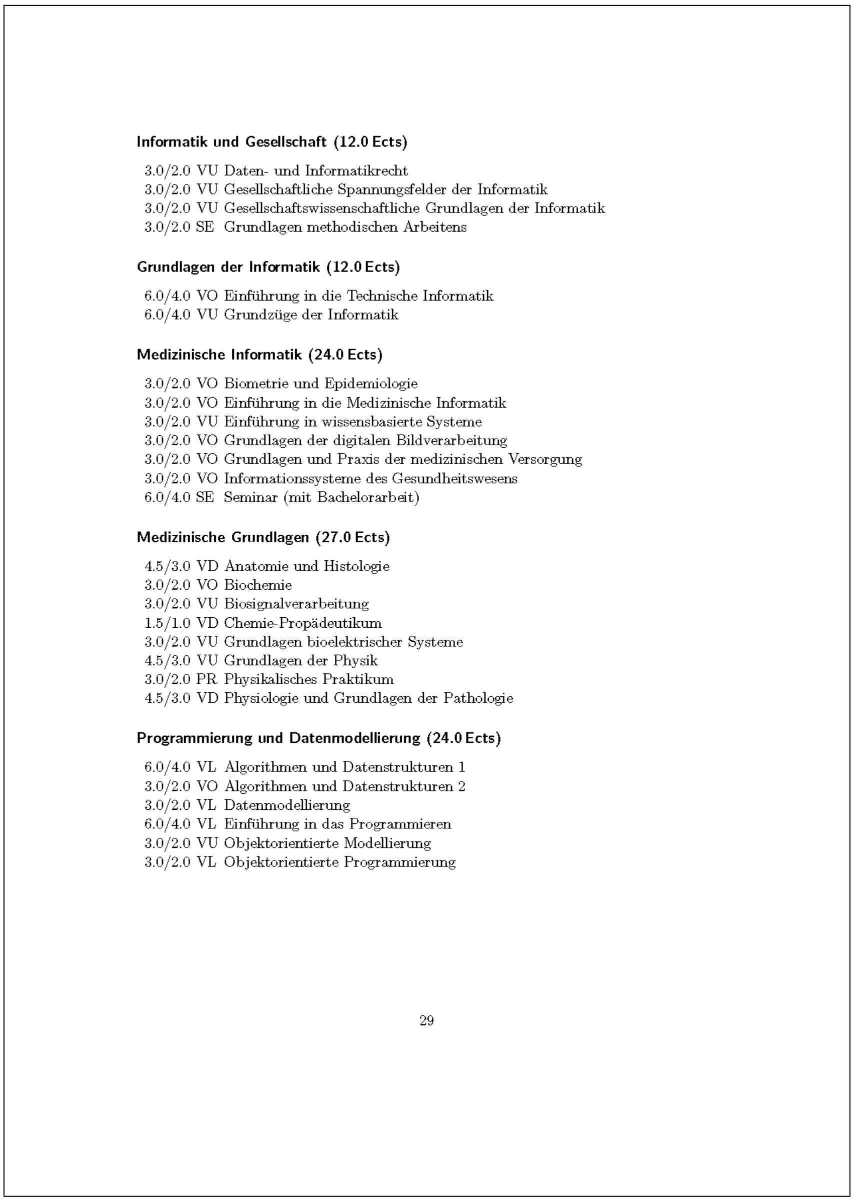
Hierzu wird als Kern des Studienganges eine solide Grundausbildung in der Informatik angeboten, mit einer Spezialisierung auf visuelle Themen wie Design, Computergraphik, Bildverarbeitung und Mustererkennung. Hier sind Gebiete wie die technische Bildaufnahme, Bildvorverarbeitung, Bildauswertung und automatische Bildinterpretation vertreten, aber auch neben Bildwiedergabe und Bildkommunikation alle Aspekte der Bildsynthese, der virtuellen Realität und der wissenschaftlichen Visualisierung.

### 3.2. Qualifikationsprofil der Absolventinnen und Absolventen

Das Studium soll eine wissenschaftlich geprägte Ausbildung vermitteln, die Theorie, Fachwissen und praktische Kenntnisse von Medientechnik, Computergraphik, der digitalen Bildverarbeitung und Mustererkennung einschließt. Es soll die Studierenden in die Lage versetzen, Methoden und Werkzeuge aus den oben genannten Gebieten zu verstehen, anzuwenden sowie sich eigenständig an ihrer Erforschung und Weiterentwicklung zu beteiligen. Studienziel ist weiters die Vermittlung von Wissen um die kreative Gestaltung der Medien und deren Produktionsprozess. Dazu gehört die Befähigung der Auszu-

20

Figure 36: Test page P3



**Informatik und Gesellschaft (12.0 Ects)**

| | | |
|---|---|---|
| 3.0/2.0 | VU | Daten- und Informatikrecht |
| 3.0/2.0 | VU | Gesellschaftliche Spannungsfelder der Informatik |
| 3.0/2.0 | VU | Gesellschaftswissenschaftliche Grundlagen der Informatik |
| 3.0/2.0 | SE | Grundlagen methodischen Arbeitens |

**Grundlagen der Informatik (12.0 Ects)**

| | | |
|---|---|---|
| 6.0/4.0 | VO | Einführung in die Technische Informatik |
| 6.0/4.0 | VU | Grundzüge der Informatik |

**Medizinische Informatik (24.0 Ects)**

| | | |
|---|---|---|
| 3.0/2.0 | VO | Biometrie und Epidemiologie |
| 3.0/2.0 | VO | Einführung in die Medizinische Informatik |
| 3.0/2.0 | VU | Einführung in wissensbasierte Systeme |
| 3.0/2.0 | VO | Grundlagen der digitalen Bildverarbeitung |
| 3.0/2.0 | VO | Grundlagen und Praxis der medizinischen Versorgung |
| 3.0/2.0 | VO | Informationssysteme des Gesundheitswesens |
| 6.0/4.0 | SE | Seminar (mit Bachelorarbeit) |

**Medizinische Grundlagen (27.0 Ects)**

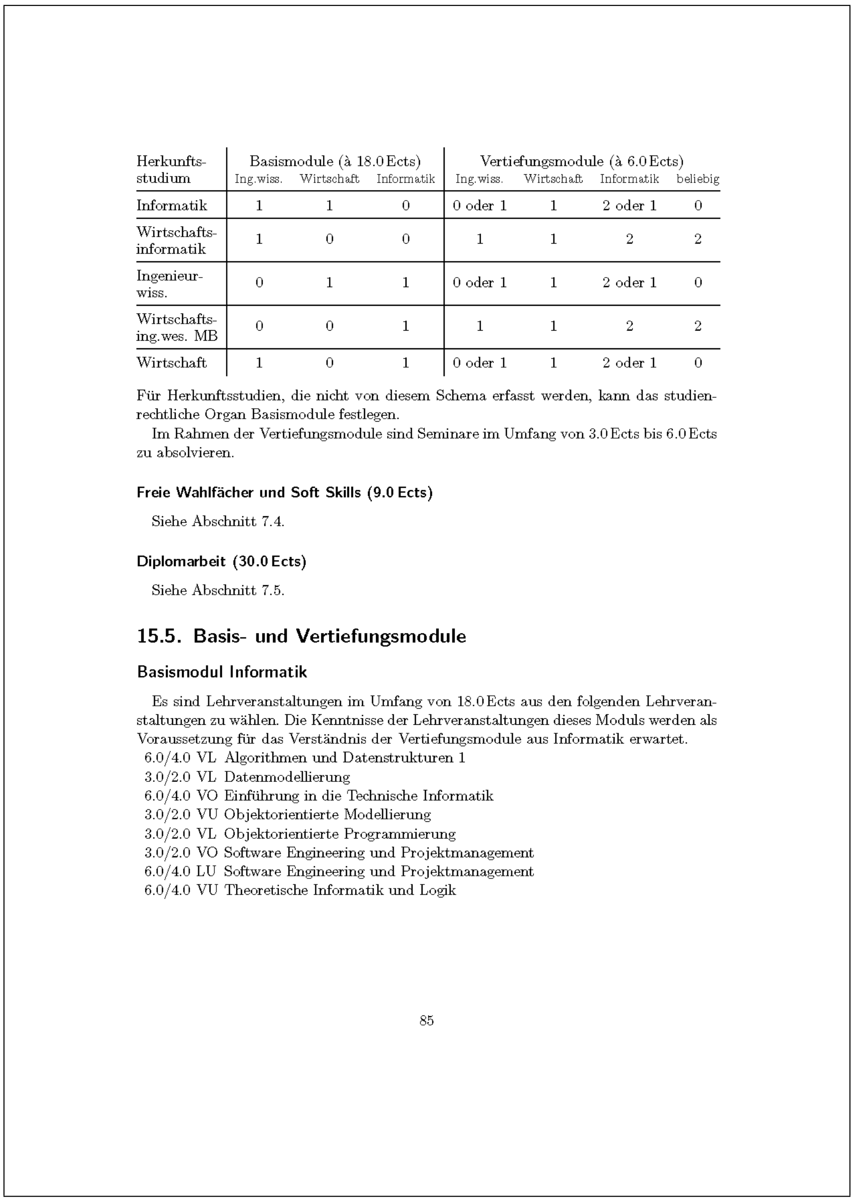
| | | |
|---|---|---|
| 4.5/3.0 | VD | Anatomie und Histologie |
| 3.0/2.0 | VO | Biochemie |
| 3.0/2.0 | VU | Biosignalverarbeitung |
| 1.5/1.0 | VD | Chemie-Propädeutikum |
| 3.0/2.0 | VU | Grundlagen bioelektrischer Systeme |
| 4.5/3.0 | VU | Grundlagen der Physik |
| 3.0/2.0 | PR | Physikalisches Praktikum |
| 4.5/3.0 | VD | Physiologie und Grundlagen der Pathologie |

**Programmierung und Datenmodellierung (24.0 Ects)**

| | | |
|---|---|---|
| 6.0/4.0 | VL | Algorithmen und Datenstrukturen 1 |
| 3.0/2.0 | VO | Algorithmen und Datenstrukturen 2 |
| 3.0/2.0 | VL | Datenmodellierung |
| 6.0/4.0 | VL | Einführung in das Programmieren |
| 3.0/2.0 | VU | Objektorientierte Modellierung |
| 3.0/2.0 | VL | Objektorientierte Programmierung |

29

Figure 37: Test page P4



| Herkunfts-studium | Basismodule (à 18.0 Ects) | | | Vertiefungsmodule (à 6.0 Ects) | | | |
|---|---|---|---|---|---|---|---|
| | Ing.wiss. | Wirtschaft | Informatik | Ing.wiss. | Wirtschaft | Informatik | beliebig |
| Informatik | 1 | 1 | 0 | 0 oder 1 | 1 | 2 oder 1 | 0 |
| Wirtschafts-informatik | 1 | 0 | 0 | 1 | 1 | 2 | 2 |
| Ingenieur-wiss. | 0 | 1 | 1 | 0 oder 1 | 1 | 2 oder 1 | 0 |
| Wirtschafts-ing.wes. MB | 0 | 0 | 1 | 1 | 1 | 2 | 2 |
| Wirtschaft | 1 | 0 | 1 | 0 oder 1 | 1 | 2 oder 1 | 0 |

Für Herkunftsstudien, die nicht von diesem Schema erfasst werden, kann das studienrechtliche Organ Basismodule festlegen.

Im Rahmen der Vertiefungsmodule sind Seminare im Umfang von 3.0 Ects bis 6.0 Ects zu absolvieren.

#### Freie Wahlfächer und Soft Skills (9.0 Ects)

Siehe Abschnitt 7.4.

#### Diplomarbeit (30.0 Ects)

Siehe Abschnitt 7.5.

### 15.5. Basis- und Vertiefungsmodule

#### Basismodul Informatik

Es sind Lehrveranstaltungen im Umfang von 18.0 Ects aus den folgenden Lehrveranstaltungen zu wählen. Die Kenntnisse der Lehrveranstaltungen dieses Moduls werden als Voraussetzung für das Verständnis der Vertiefungsmodule aus Informatik erwartet.

| | | |
|---|---|---|
| 6.0/4.0 | VL | Algorithmen und Datenstrukturen 1 |
| 3.0/2.0 | VL | Datenmodellierung |
| 6.0/4.0 | VO | Einführung in die Technische Informatik |
| 3.0/2.0 | VU | Objektorientierte Modellierung |
| 3.0/2.0 | VL | Objektorientierte Programmierung |
| 3.0/2.0 | VO | Software Engineering und Projektmanagement |
| 6.0/4.0 | LU | Software Engineering und Projektmanagement |
| 6.0/4.0 | VU | Theoretische Informatik und Logik |

85

Figure 38: Test page P5

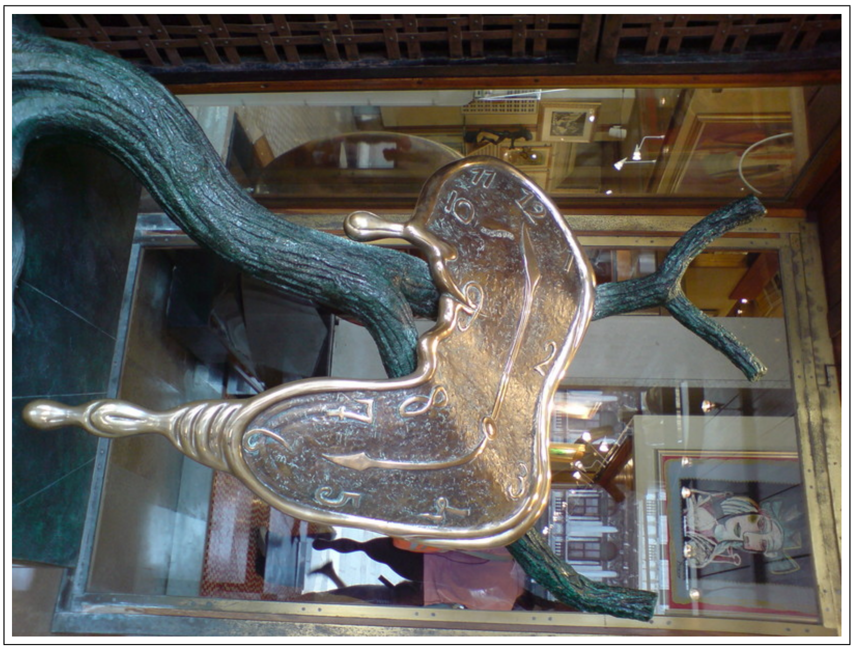


Figure 39: Test page Q1

## References

[1] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton Unversity Press, 2006.

[2] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA J. Comput.*, 1994.

[3] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 2003.

[4] J. Brassil. Tracing the source of a shredded document. Technical report, HP Laboratories, 2002.

[5] R. E. Burkard, E. Cela, P. M. Pardalos, and L. S. Pitsoulis. The quadratic assignment problem. *European Journal of Operational Research*, 1998.

[6] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, 1976.

[7] J. Cirasella, D. S. Johnson, L. A. McGeoch, and W. Zhang. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. *Lecture Notes in Computer Science*, 2153, 2001.

[8] E. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

[9] M. Dorigo and G. D. Caro. The ant colony optimization meta-heuristic. *New Ideas in Optimization*, 1999.

[10] F. Glover and M. Laguna. Tabu search. *Kluwer Academic Publishers*, 1997.

[11] R. Gonzalez and R. Woods. *Thresholding. In Digital Image Processing.* Pearson Education, 2002.

[12] R. C. Gonzalez and R. E.Woods. *Digital Image Processing.* Addison-Wesley, 1992.

[13] P. Hansen and N. Mladenovic. A tutorial on variable neighborhood search.

[14] E. Justino, L. S. Oliveira, and C. Freitas. Reconstructing shredded documents through feature matching. *Forensic Science International*, 160:140–147, 2006.

[15] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 1983.

[16] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag, 1996.

[17] N. Mladenović and P. Hansen. Variable neighborhood search. *Comps. in Opns. Res.*, 24:1097–1100, 1997.

[18] C. Papaodysseus, T. Panagopoulos, M. Exarhos, C. Triantafillou, and D. Fragoulis. Contour-shape based reconstruction of fragmented, 1600 b.c. wall paintings. *IEEE Transactions on Signal Processing*, 50, 2002.

[19] T. Sikora. The MPEG-7 visual standard for content description - an overview. *IEEE Transactions on Circuits and Systems for Video Technology*, 11, 2001.

[20] A. Skeoch. *An Investigation into Automated Shredded Document Reconstruction using Heuristic Search Algorithms.* PhD thesis, University of Bath, 2006.

[21] E. Skoudis. *Counter hack: a step-by-step guide to computer attacks and effective defenses.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[22] A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22, 2000.

[23] T. Stutzle. Iterated local search for the quadratic assignment problem, 1999.

[24] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parall. Comput.*, 17, 1991.

[25] A. Ukovich and G. Ramponi. Features for the reconstruction of shredded notebook paper. In *ICIP (3)*, pages 93–96, 2005.

[26] A. Ukovich, G. Ramponi, H. Doulaverakis, Y. Kompatsiaris, and M. Strintzis. Shredded document reconstruction using MPEG-7 standard descriptors. *IEEE Int. Symp. on Signal Processing and Information Technology*, pages 18–21, 2004.

[27] A. Ukovich, A. Zacchigna, G. Ramponi, and G. Schoier. Using clustering for document reconstruction. In E. R. Dougherty, J. T. Astola, K. O. Egiazarian, N. M. Nasrabadi, and S. A. Rizvi, editors, *Image Processing: Algorithms and Systems, Neural Networks, and Machine Learning*, volume 6064 of *Proceedings of SPIE*. International Society for Optical Engineering, February 2006.

[28] R. Vaessens, E. Aarts, and J. Lenstra. Job-shop scheduling by local search, 1994.

[29] H. Wolfson. On curve matching. *IEEE Trans. Pattern Anal. Mach. Intel.*, 12, 1990.