FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Reconstructing Cross-Cut Shredded Documents by means of Evolutionary Algorithms

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Christian Schauer
Matrikelnummer 9925377

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Matthias Prandtstetter

Wien, 17.05.2010

(Unterschrift Verfasser)

(Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit — einschließlich Tabellen, Karten und Abbildungen —, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Ort, Datum	Unterschrift

Abstract

This master thesis focuses on the reconstruction of destroyed text documents, which were mechanically destructed with the help of so-called cross-cut shredders. These machines cut a piece of paper into equally sized, usually rectangular, slices, which leads to the fact that these shreds are indistinguishable on the basis of their shape. The ambition of this master thesis is to automatically reconstruct cross-cut shredded text documents true to original. To fulfil this aim a genetic algorithm (GA) has been developed, implemented and tested.

At the beginning a formal definition of this problem and references to related work will be given. While there are some approaches published dealing with the reconstruction of destroyed paper in general, there is barely work done in the field of cross-cut shredding. An introduction to the working principle of GAs as well as other mainstreams of evolutionary computation will be given.

The considered problem obviously also has two-dimensional geometric aspects. Due to the fact that the most popular GA operators were designed to work on one-dimensional solution representations, some proved operators were adapted and others newly created to match the needs of this master thesis.

To further improve the GA a local search phase based on variable neighbourhood search (VNS) is embedded.

Finally computational results for ninety instances based on ten different documents are presented, which were used for evaluating the designed operators. It could be shown that one of the presented approaches outperforms previously described methods based on ant colony optimisation and VNS only.

Zusammenfassung

Der Fokus dieser Masterarbeit liegt auf der Rekonstruktion von zerstörten Textdokumenten, die durch den maschinellen Einsatz von sogenannten Cross-Cut Schreddern vernichtet wurden. Diese Geräte schneiden das Papier in regelmäßige, gleichgroße — bevorzugterweise rechteckige — Teile, was dazu führt, dass diese Schnipsel anhand ihrer Form nicht mehr unterscheidbar sind. Das Bestreben dieser Masterarbeit ist nun maschinell durch Cross-Cut Schredder vernichtete Dokumente originalgetreu wiederherzustellen. Hierfür wurde ein Genetischer Algorithmus (GA) entwickelt, implementiert und getestet um sich dieser Herausforderung zu stellen.

Zu allererst wird aber eine formale Definition dieses Problems gegeben und auf verwandte Themen verwiesen. Während nämlich für die Papierrekonstruktion an sich ein paar wenige Ansätze bereits publiziert wurden, ist das Gebiet rund um den Cross-Cut Schredder noch kaum erschlossen. Weiters wird eine Einführung in das Funktionsprinzip von GAs und darüber hinaus in die anderen Gebiete der Evolutionären Algorithmen gegeben.

Das behandelte Problem bezieht sich, im geometrischen Sinne, auf einen zweidimensional Raum. Da die ausgereiften GA Operatoren aber auf eindimensionalen Lösungsrepräsentationen arbeiten, mussten für diese Masterarbeit bewährte Operatoren adaptiert und neue entworfen werden.

Um den GA noch weiter zu verbessern wurde eine lokale Suche mittels variabler Nachbarschaftssuche (VNS) eingebunden.

Schlussendlich werden die Testergebnisse von neunzig Instanzen basierend auf zehn Dokumenten präsentiert, wobei diese Resultate zur Evaluation der erstellten Operatoren dienen. Einer dieser Ansätze war nachweisbar im Stande die bisherigen Methoden aufbauend auf Ameisenkolonie Optimierung und alleiniger VNS zu schlagen.

Danksagung

An dieser Stelle möchte ich die Möglichkeit nutzen mich bei allen Personen zu bedanken, die zum Gelingen dieser Arbeit und dem damit verbundenen Abschluss meines Studiums beigetragen haben.

Zu allererst möchte ich Günther Raidl danken, dass er mich vor vielen Jahren in das spannende Gebiet der Evolutionären Algorithmen eingeführt hat und mir nun die Möglichkeit geboten hat, das bei ihm Erlernte im Bereich der Dokumentenrekonstruktion anzuwenden.

Mein ganz besonderer Dank gilt Matthias Prandtstetter, der mir unermüdlich in Zeiten des Fortschrittes mit Rat und Tat zur Seite gestanden ist, während er in Phasen der Stagnation Geduld walten ließ.

Weiters möchte ich mich bei meinen Freunden bedanken, die mir sowohl für fachdienliche Diskussionen als auch für entspannende Zerstreuung stets zur Seite standen.

Zu allerletzt möchte ich meiner Familie im Allgemeinen und meinen Eltern im Speziellen danken, von denen ich während meines gesamten bisherigen Bildungsweges immer unterstützt wurde.

Contents

CHAPTER 1

INTRODUCTION

In his utopian novel *Nineteen Eighty-Four* George Orwell (1903–1950) presented the perfect way to devour a document. Not only that the paper itself is dropped into the so-called *memory hole* to be burned there, moreover all cross-references to the document are deleted and thus its existence *vaporized*, as Orwell named it. The complete description of vaporization and the public system enabling this process is given here [26].

While the year 1984 is long gone, Orwell's suggestion still remains unimplemented so far. Nowadays the device that resembles the memory hole the most—maybe not in performance but at least in distribution—is the so-called shredder.

Despite the digitalisation of our world most documents are yet printed on paper, sometimes because of legal reasons. On the other hand these papers have to be destroyed sufficiently when not needed anymore, also because of legal reasons.

In certain situations might arise the need to reconstruct the—either manually or mechanically—destroyed documents for any reasons whatsoever. One of the most famous examples of document reconstruction is the case of the *"Stasi Akten"*. The *Ministry for State Security*, colloquial *Stasi*, was the official state security service of the communistic *German Democratic Republic*. Before the reunification with the *Federal Republic of Germany* in 1990 the Stasi destroyed great parts of its printed database. This resulted in 16000 bags containing 600 million snippets either torn apart or shredded. In more than ten years the content of only 250 bags were manually restored by 15 employees. To help them a computer aided reconstruction program was commissioned [1].

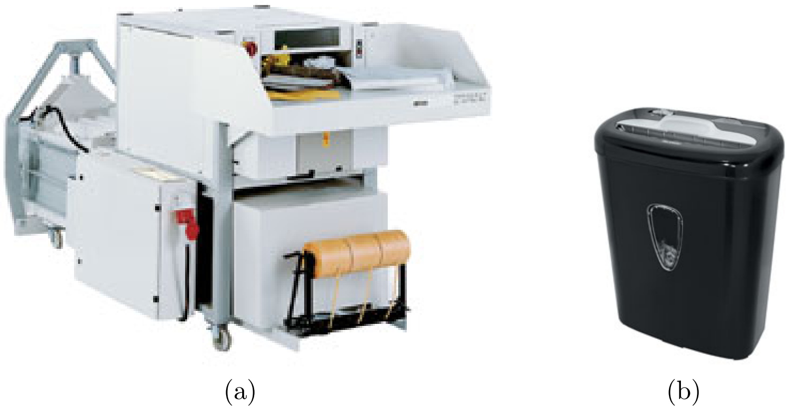


Figure 1.1: (a) Shows the Ativa V5088C industrial shredder for about 60.000$ (b) Shows the Ativa LD100 home office shredder for 50$ both taken from [3]

The easiest way to make a sheet of paper unreadable would be to tear it apart but this makes every piece of it unique because the paper edges can be distinguished through their different shape. They are frayed and feature shearing effects. In [28] the rebuilding problem is called the *reconstruction of manually torn paper documents* (RMTPD) and this term will also be used here.

A better safety degree can be gained by using a mechanical device—the shredder. This gadget uses against each other rotating knives through which the paper is passed, which results in snippets of the same shape and size. Because of the cutting process the edges are flat and no additional shape information can be retrieved on the borders of the snippets.

Therefore all information that can help to rebuild the original document, must be extracted from the inner regions of the snippets, i.e., only the printed texture and the way this structure was cut apart. This directly leads to a differentiation between text documents and printed pictures. Because of the entire different structure of these two types, different approaches for the information retrieval will promise satisfying results. This master thesis will only discuss the reconstruction of text documents.

The basic model of shredders, the *strip shredders*, cut the sheets along their whole length and thus producing so-called *strips*, which are as long as the original sheet of paper see Fig. 2.1a. Again the term from [28], which is *reconstruction of strip shredded text documents* (RSSTD), will be used within this master thesis.

More complex and thus more secure shredders also cut along the width and therefore create snippets or *shreds* that are smaller than the original document's height,

Table 1.1: DIN standard EN 15713 for shredders according to [2]

security level	**average area** in mm^2	**max. cutting width** in mm
1	5000	25
2	3600	60
3	2800	16
4	2000	12
5	800	6
6	320	4

see Fig. 2.1b and Fig. 2.1c, respectively. These shreds are the product of the so-called *cross-cut shredders.* This problem is denoted as the *reconstruction of cross cut shredded text documents* (RCCSTD) in [28]. Overall the German Institute for Standardisation (DIN) defines a standard for shredders differentiating between six security levels, see Tab. 1.1.

As already hinted it is necessary to gain as much information from the snippets as possible to reconstruct the original document. This directly leads to pattern recognition approaches to collect these information.

Overall the snippets must be digitised first to apply these methods. Again different techniques, which work on the digitised snippets, are used for printed text and pictures. For pictures the so-called *content-based image retrieval* collects information like colour or texture [13]. While this technique still is in its infancy, the *optical character recognition* (OCR) is a technically mature approach to extract letters from a text document [27].

However an implementation of an OCR approach is as complex as time-consuming and thus beyond the scope of this master thesis. Moreover the focus of this thesis lies on the combinatorial optimisation methods to rebuild a document, after the pattern recognition was done. Therefore, to gain information from the snippets, a quite simple but nonetheless efficient approach was used in a way that it can easily be expanded. In the end a reconstruction system must of course contain an information retrieval approach as well as a combinatorial optimisation method to process these data.

In the next section a formal defintion of the problem and references to related work will be given. Afterwards the *The Theorie of Evolution* and the main working principles of *Evolutionary Algorithms* will be discussed, before the, for this master thesis designed, algorithm is presented.

CHAPTER 2

Problem Definition

Due to the lack of shape information helping to restore the shredded documents it is convenient to choose a combinatorial approach as done for this master thesis. For this purpose a formal definition of the problem will be given in the following. Although slightly extended and redefined the nomenclature and definition are based on [28].

Initiatively the following properties of the shreds need to be assumed. To ensure that the algorithm later presented will work correctly, only shredding devices will be considered, which cut the whole page along a horizontal and vertical grid, as visualised in Fig. 2.1b. This leads to rectangular shreds that are all of the same height and width. While duplex print is a common feature for office printers, it is not considered here. Therefore all the shreds only contain information on one side.

Furthermore must be considered that in some situations no statement about the correct position of shreds can be made at all; for example consider a text written in columns and then cut right between them. While there is a good chance that each fragment can be reconstructed properly, the correct order of the columns can only be restored from the document's context. Another example, which often occurs in cross-cut shredding, is a horizontal cut between two lines. Again without the context a complete reconstruction is beyond the scope of the current computer systems.

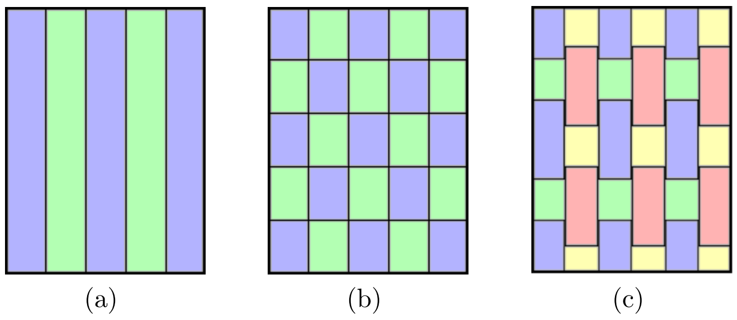


Figure 2.1: Three different cutting patterns

2.1 Formal Problem Definition

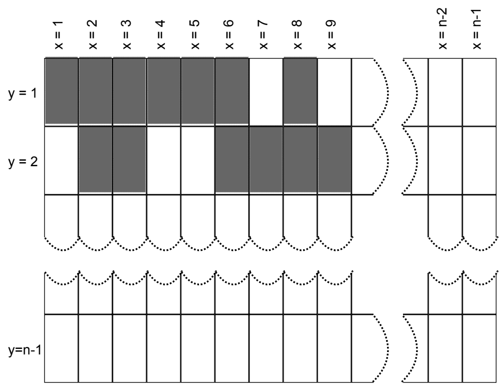
Let $S = \{1, \ldots, n\}$ be a set of rectangular, geometrically identical shreds that are the result of a shredding device with the above mentioned characteristics. While the shreds $1, \ldots, n-1$ contain the text of the original document, all the blank shreds are combined to a single special shred n, which will be called the *virtual shred.* This is mainly done for modelling reasons. There is to note that no useful information can be extracted from completely empty shreds. While shreds $1, \ldots, n-1$ must appear exactly once, the virtual shred might be utilised several times, e.g. for filling empty regions.

Furthermore it is assumed that the correct orientation of each shred is already known, e.g, by determing them in a pre-processing step, and therefore no operation, like rotation, needs to be considered in the further context. A pattern recognition technique, for example described in [7], could be used to realise this pre-processing.

Obviously it is the goal to minimise the overall error made during the reconstruction process, i.e., placing only these shreds next to each other, which were *neighbours* in the original document. For estimating their error when placing two shreds $i, j \in S$ next to each other two functions are needed for the formal definition; see Sec. 2.2 for the *error estimation function* $c_h(i, j)$ and $c_w(i, j)$, which computes the error made when i is left or on top of j, respectively.

A solution of the *reconstruction of cross cut shredded text documents* (RCCSTD) problem is defined as an injective map $\Pi : S \setminus \{n\} \to \mathbb{D}^2$ of shred positions $p = (x, y)$ in the two-dimensional (Euclidean) space $\mathbb{D}^2$, with $x, y \in \mathbb{D} = \{1, \ldots, n-1\}$, such that every space is filled with one shred at most and each shred $i \in S \setminus \{n\}$ occurs exactly once, while the virtual shred is assigned to the remaining empty spaces, i.e., n acts as a placeholder. See Fig. 2.2b for a visualisation.

(a)



(b)

Figure 2.2: (a) A bunch of cross-cut shreds that awaits to be reconstructed
(b) Drawing of a solution; shreds $\{1, \ldots, n-1\}$ are grey; the virtual shred is white

Moreover the function $s(p)$ returns the shred at position p:

$$s(p) = \begin{cases} i & \text{if there is a shred } i \in S \text{ such that } \Pi(i) = p \\ n & \text{otherwise} \end{cases}, \quad \forall p \in \mathbb{D}_0^2,$$

with $\mathbb{D}_0 = \{0, \ldots, n\}$. To enhance the space $\mathbb{D}^2$ to $\mathbb{D}_0^2$ means that the whole solution is surrounded by virtual shreds, which is again done for modelling reasons.

On the basis of $s(p)$ with $p = (x, y) \in \mathbb{D}^2$ the four neighbours of a shred can be defined as:

$$s_l \mathrel{\widehat{=}} s((x-1), y) \qquad s_t \mathrel{\widehat{=}} s(x, (y-1))$$
$$s_r \mathrel{\widehat{=}} s((x+1, y)) \qquad s_b \mathrel{\widehat{=}} s(x, (y+1))$$

The aim of RCCSTD is now to find an assignment of shreds to positions within a solution such that the overall error induced by all realised neighbourhoods is minimised, which leads to the following cost function:

$$c(\Pi) = \sum_{p \in \{1,\ldots,n\}^2} c_h(s_l(p), s(p)) + c_w(s_t(p), s(p)). \tag{2.1}$$

At first sight it can be seen that this representation barely sets a limit to the size of the solution. On the other hand allows this attempt that well matching sequences of shreds can stay together and are not forced to be torn apart at the end of a row or column of limited capacity, which would also contradict the *building block hypothesis* from Sec. 3.2.1.

2.2 Error Estimation Function

How good do two shreds fit together if sited next to each other?

The answer to this question can be found in the *error estimation function*, which deals with the problem to evaluate how good or bad the shreds i and j match together.

Obviously there are multiple approaches to define such a function. Common to all of them is that none will work perfectly over all problems. A discussion about these approaches and their drawbacks can be found in [28]. In the following only the function, which was used within this master thesis, will be introduced.

Given a set $S = \{1, \ldots, n\}$ of shreds, having the characteristics mentioned above, the error estimation function will compute the quality of two neighbours $i, j \in S$.

The idea is now to ignore the features of the inner regions and to focus only on the (colour) information along the edges. Considering that all the shreds are digitised by a scanner the information to deal with is represented as an image with a previously defined resolution. Hence the number of (image) pixels along the y-axis of a shred $i \in S$ will be called h_i while the number of pixels along the x-axis will be w_i.

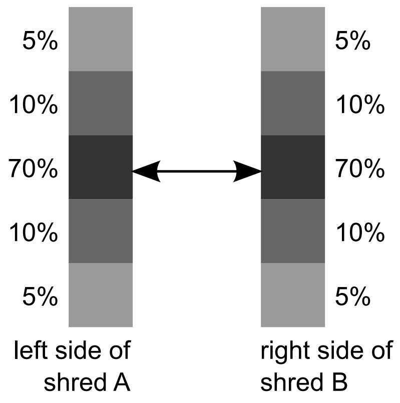
Due to the fact that all shreds feature the same size and shape it is likely that the digitised snippets are all of the same resolution and therefore it is assumed that:

$$h_i = h_j \quad \wedge \quad w_i = w_j \qquad\qquad \forall i, j \in S$$

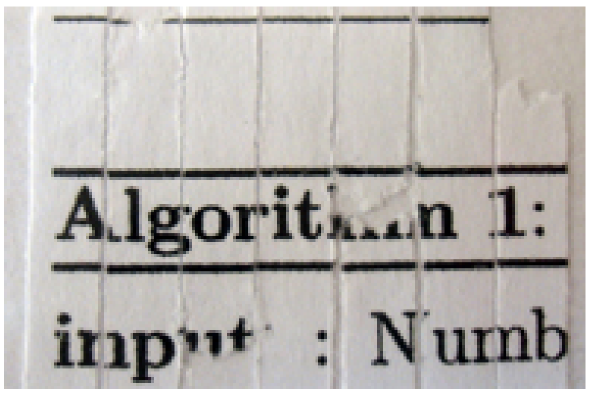
thus for convenience the indices will be omitted and h will always refer to the height while w stands for the width.

While most documents are printed in black and white, it seems sufficient to work with the B/W colour space, which indeed proved efficient for the *reconstruction of strip shredded text documents* (RSSTD) problem—see [28]. But the RCCSTD shreds are usually smaller than the RSSTD strips, thus less information is contained along the edges. Moreover preliminary tests have shown that the expansion to the 8-bit grayscale colour space pays off and builds therefore the base for the error estimation function.

When shred i is left of j then the right edge of i and the left edge of j lie opposite to each other. Otherwise when i is on top of j then the bottom edge of i touches the top edge of j. Knowing now that all shreds are of the same size and thus resolution, the edges of two shreds can be compared simply by comparing the opposite pixels of the edges. To smooth the side-effects of rasterisation two pixels above and below the current position will also be considered in a weighted sum following the idea taken from [6].



(a)



(b)

Figure 2.3: (a) Visualises the weighted average of two opposite pixels taking their neighbours into account; (b) A small reconstructed document, note the frayed edge around the “hm“

As already mentioned two functions are needed: one for the relation between left and right $c_h(i,j)$ and the other $c_w(i,j)$, for the top and bottom relations, with $i,j \in S$.

Obviously the best case for both $c_h(i,j)$ and $c_w(i,j)$ would be two perfectly fitting shreds. Which means no error will occur at all. Thus this case should lead to an error equal zero. Moreover the value of $c_h(i,j)$ and $c_w(i,j)$ should increase the more error is made, which leads to the following requirement:

$$\begin{aligned} c_h(i,j) &\geq 0 \\ c_w(i,j) &\geq 0 \end{aligned} \qquad \forall i,j \in S.$$

Furthermore we assume that the grayscale value of a pixel can be retrieved by the functions:

$$\begin{aligned} v_l(i,y), v_r(i,y) &\in \{0,\dots,255\} && \text{with } 1 \leq y \leq h \text{ for the left and right edge} \\ v_t(i,x), v_b(i,x) &\in \{0,\dots,255\} && \text{with } 1 \leq x \leq w \text{ for the top and bottom edge} \end{aligned}$$

of each shred $i \in S$.

The main idea of the error estimation is to compare the weighted average of the five grayscale pixels of shred i with the corresponding average value of shred j along the x-axis or the y-axis and build the sum over all errors along an edge. A visualisation of this weighting can be seen in Fig. 2.3a.

This leads to the formal definition of $c_h(i,j)$:

$$c_h(i,j) = \sum_{y=3}^{h-2} e_h(i,j,y) \tag{2.2}$$

$$e_h(i,j,y) = \begin{cases} 1 & \text{if } e'_h(i,j,y) \geq \tau \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

$$\begin{aligned} e'_h(i,j,y) = \Big| & 0.7 \cdot \big(v_r(i,y) - v_l(j,y)\big) \\ & + 0.1 \cdot \big(v_r(i,y+1) - v_l(j,y+1)\big) \\ & + 0.1 \cdot \big(v_r(i,y-1) - v_l(j,y-1)\big) \\ & + 0.05 \cdot \big(v_r(i,y+2) - v_l(j,y+2)\big) \\ & + 0.05 \cdot \big(v_r(i,y-2) - v_l(j,y-2)\big)\Big| \end{aligned} \tag{2.4}$$

and $c_w(i,j)$:

$$c_w(i,j) = \sum_{x=3}^{w-2} e_w(i,j,x) \tag{2.5}$$

$$e_w(i,j,x) = \begin{cases} 1 & \text{if } e'_w(i,j,x) \geq \tau \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

$$\begin{aligned} e'_w(i,j,x) = \Big| & 0.7 \cdot \big(v_t(i,x) - v_b(j,x)\big) \\ & + 0.1 \cdot \big(v_t(i,x+1) - v_b(j,x+1)\big) \\ & + 0.1 \cdot \big(v_t(i,x-1) - v_b(j,x-1)\big) \\ & + 0.05 \cdot \big(v_t(i,x+2) - v_b(j,x+2)\big) \\ & + 0.05 \cdot \big(v_t(i,x-2) - v_b(j,x-2)\big)\Big| \end{aligned} \tag{2.7}$$

The pixel weighting factors and the threshold τ were chosen through a process of preliminary tests. The best results were reached when setting $\tau = 25$, which means that a difference of more than 10% of grayscale shades will result in an error.

2.3 About the Complexity

Drawing attention to the complexity of a problem like this means to disucss how much can be anticipated from an algorithm. In computer science there is a set of so-called $\mathcal{NP}$-complete problems. One of the most famous is the *travelling salesman problem* defined in [20].

These problems have in common that till now no polynomial algorithm is known that can always solve one these problems in an optimal manner. The insteresting fact is that if an algorithm is found for one problem it can be mapped to all the other problems in a way that it would still work in polynomial time. Nowadays only exponential algorithms could optimally solve $\mathcal{NP}$-complete problems but for larger instances computing a solution with these algorithms is beyond the scope of current computer technologies. Thus the best approach yet is to design specific polynomial algorithms for each problem that approximate as best as possible to the optimal solution.

In [28] the RSSTD is directly mapped to the (symmetric) travelling salesman problem and thus RSSTD proves to be $\mathcal{NP}$-complete. Furthermore there is to note that strip shredding is a special case of cross cut shredding, there are no vertical cuts. Therefore RCCSTD is at least as complex to solve as RSSTD and thus RCCSTD is $\mathcal{NP}$-hard, which means that it is at least as hard to solve as the $\mathcal{NP}$-complete problem of RSSTD.

2.4 Related Work

The procedure of document reconstruction reminds a lot of playing with a jigsaw puzzle. Because the latter is a game, thus the rules are conceived to have fun playing it. Therefore two requirements are made for every jigsaw. In a high quality jigsaw puzzle the pieces fit perfectly into another, which makes the result singular. Moreover the result is always announced on the packing and so the information printed on the pieces also helps to identify each piece's position. These two assumptions and some handwork will lead to a finished jigsaw. When on the other hand the computer should solve one, then some related work can be found in [4, 8, 9, 14, 38].

Reconstructing destroyed documents is done for serious reasons and would not be necessary if the result is already known, which destroys the second assumption. While hand torn snippets might be unique in shape it is not mandatory. Moreover because of frayed edges it is likely that two snippets would not perfectly fit together. How to deal with this problem and others arising with hand torn paper are discussed in [11, 18, 28].

Mechanically destroyed paper snippets look all the same, which flattens the first requirement for jigsaw puzzles. Thus another method is needed to find out if two snippets fit together, like the *error estimation function* presented in Sec. 2.2.

These methods are in some way the main commonness between working on a jigsaw puzzle and reconstructing destroyed documents because the human brain owns excellent abilities in the fields of pattern recognition and image processing, which are used when working with jigsaw pieces to match to edges or to identify the imprinted image. Therefore it is obvious to use computer vision techniques for an error estimation function.

To extract printed letters the field of *optical character recognition* offers various mature methods like described in [27]. But very often the characters on the edges are cut apart and not identifiable and thus a stronger approach for information retrieval is needed. This directly leads to *image processing* and the until now infantile but fast developing field of *content-based image retrieval.*

These systems are used to gain and store information from an image to manage a database of images. Museums, medical image management, multimedia libraries and document archives are some examples where these applications are used. The indexing and retrieval was usually achieved by a text-based approach, which means a text description of either key words or free text for each image. While this approach is subjective and some visual information cannot be described at all, an alternative form of description relies on the inherent properties of the images. The idea is to use data like patterns, colours, textures, shapes of image objects and their related layout and location information to extract standardised information and store it in the database. How to extract these data and deal with it can be found in [13].

An implementation of the previously mentioned kind of content description is for instance realised in the *MPEG-7* standard, which is introduced in [22]. Ukovich *et al.* used these MPEG-7 descriptors in [35] in the context of the strip shredding problem. They expanded their list of extracted features with characteristics especially related to text documents like line spacing or text colour in [34]. Then they used all the collected information from many sheets of paper and designed a clustering algorithm in [36]. This algorithm builds groups or clusters of strips with equal features. Taking into account that each sheet of paper differs a little bit in its design from the others, these groups can be seen as a way to differentiate the strips and assign them to one of the orignal sheets. This represents an interesting approach of pre-processing because now only each page has to be reconstructed on its own and not all permuations of all shreds must be considered at once.

Another problem is to find the correct orientation of the shreds. A shred that is rotated by 180° might fit well at a certain position, according to the error estimation function, but the position of the shred is wrong anyway. As already mentioned this problem is already solved by definition for this master thesis. Of course the shreds do not automatically orientate correctly but a procedure to solve this problem is needed as described in [5] or [21].

All the related work mentioned till now does in fact not deal with the RCCSTD or cross-cut shredding in general but with related problems like hand torn or strip shredded paper. The only work, to the best knowledge, about RCCSTD is done by Prandtstetter and published in [28] and [29]. In [28] various algorithms are presented to solve the RCCSTD like an *ant colony optimisation*, a *variable neighbourhood search*, *based on integer linear programming techniques* and several constructions heuristics. Moreover sample instances were provided to demonstrate the quality of the algorithms. In [29] a vast set of test results is presented using the instances previously mentioned. These results and the underlying instances will be used as comparison for the test results of this thesis.

CHAPTER 3

Evolutionary Algorithms

In this master thesis the previously defined problem of document reconstruction is solved by means of an evolutionary algorithm (EA). This kind of metaheuristic is inspired by the biological evolution and thus makes it one representative of a bionic approach in computer science.

Before the working principles and different classes of EAs are described a short historical side trip through the theses about evolution will be made, while a more detailed abstract to that theme can be found in [17]. Furthermore an introduction to genetics and evolutionary factors will be given, but which is limited to the terminology of EAs only, while the wide field of these aspects is satisfyingly discussed in [12].

3.1 (Re|E)volution

In the 17*th* century James Ussher(1581–1656) the archbichop of Armagh assessed the creation of our world according to the bible on 23 October 4004 BC at 6:00 pm. This calculation was so accurate that in 1701 the church of England accepted this date and published it in the introduction of the King-James-Bible till the beginning of the 20th century.

In the 18*th* century Carl von Linné (1707–1778) published the *Systema naturae* a complete encyclopaedia containing all then known 4000 animals and 14000 plants. Linné assumed that all kinds find their beginning in an act of creation and from that moment on no other life forms will be created nor existing kinds will extinguish.

Since these works were pulished the picture of our world and universe has changed a lot. The universe aged from thousands of years—according to Ussher—to billions. Therefore life had enough time to evolve in different directions and nowadays we know millions of different kinds. So for better understanding of the *Theorie of Evolution* a short historical survey will be given here.

3.1.1 The History of Revolution

The idea of an evolutionary movement, that enabled the multiple forms of life on earth out of some primitive kinds, can first be found at the beginning of the 19*th* century. Charles Darwin's grandfather Erasmus Darwin established such a theory which later profoundly affected the zoologist Jean Baptiste de Lamarck.

It is comprehensible that in the 18*th* century Linné's idea of the constancy of all kinds seemed rational. The change that affect life forms happens in millions of years, a period that was then beyond imagination. Not only that variations between following generations are barely recognisable, there are also great gaps between the fossil materials that provide information about the evolutionary process. But by finding more and more fossils scientists were forced to rethink the theory of the constancy of kinds due to huge contradictions. How was it for instance possible to find fossils of sea dwellers in alpine regions?

It was Georges Baron de Cuvier (1769–1832) the founder of palaeontology who proved at the end of the 18*th* century the extinction of old and the creation of new species by interpreting fossil discoveries, which led Cuvier to his *Catastrophism.* This theory assumed that by natural catastrophes all kinds—local or global—were extinguished. After a certain disaster nature was reset and new kinds were created, which would not alter until the next catastrophe.

After studying various kinds of animals and plants Jean Baptiste de Lamarck (1744–1829) recognised the astounding ability of all life forms to adjust to the surrounding world in a seemingly optimal manner. In 1809 Lamarck described in his work *Philosophie zoologique* a complete theory of evolution that assumed four principles:

Adaptive Force allows each organism to adjust to its surrounding environment. Due to Lamarck each kind will improve until the limit of development is attained.

Spontaneous Generation is the origin of the variation of kinds. In the face of the *adaptive force* the question arises of a mechanism that explains the existence of all the various species that are well adapted to their environment. The *spontaneous generation*—which resembles nowadays the mutation—is responsible for the creation of new kinds in the focus of their environment.

Complexifying Force helps each kind to better adjust to the general living conditions. According to Lamarck the necessity of a species is responsible for the development of certain characteristics. Lamarck saw a desire to perfection in nature itself, which helps all kinds to evolve the needed attributes.

Soft Inheritance allows parents to pass acquired characteristics and abilities as well as already inherited features on to their next descendants.

At the end of the 19*th* century Lamarck's theory was abandoned. Especially the *soft inheritance* was not maintainable any more. But while in previous theories always the whole kind was considered as a continuous being, it was Lamarck who first set the individual in the focus of evolution. The individual became the driving factor in the development of each kind. This point also became one of the main principles of Darwin's *Theory of Evolution.*

3.1.2 The Theory of Evolution

An advancement to Lamarck's theses was developed by Charles Darwin (1809–1882). Darwin's *Theory of Evolution* builds the fundament of our present evolutionary biology. While Lamarck was focused on the temporal development of populations, Darwin now deals with the endless variation of different kinds.

Darwin's probably best known book is *On the Origin of Species by Means of Natural Selection.* One important approach of his work was that human kind is part of the evolutionary process and therefor the *Homo Sapiens* loses any special status and becomes just another part of nature, which Darwin describes in *The Descent of Man.* His work in general does not present a single theory about evolution but a bunch of theses that deal with the problem. So the main themes of his theory can be summarised as the following:

Evolution Itself Darwin sees the world not as a statical but as a dynamical system. Especially the flora and fauna changes because of the creation of new and the extinction of old kinds. This point is the complete antagonism of Linné's picture of the world and life forms.

Combined Evolution Akin organisms share the same ancestors. In the end all groups of organisms find their derivation in a single form of life.

Multiplication of Kinds The huge variation of kinds is explained by the idea that every species derives to various new kinds.

Gradualism The evolutionary process happens slowly throughout gradual changes of the populations. The change performs continuous over a long period of time without sudden incoherent alterations.

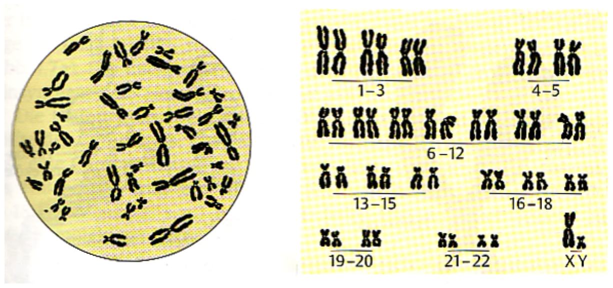


Figure 3.1: Genome of a human cell, taken from [12]

Natural Selection Important for a long term existence of a population is a great set of genetical variations. The small amount of individuals that are very well adapted to the environment will survive and build the base of the following generations. This means that the terms of nature will select the parents because of their survival.

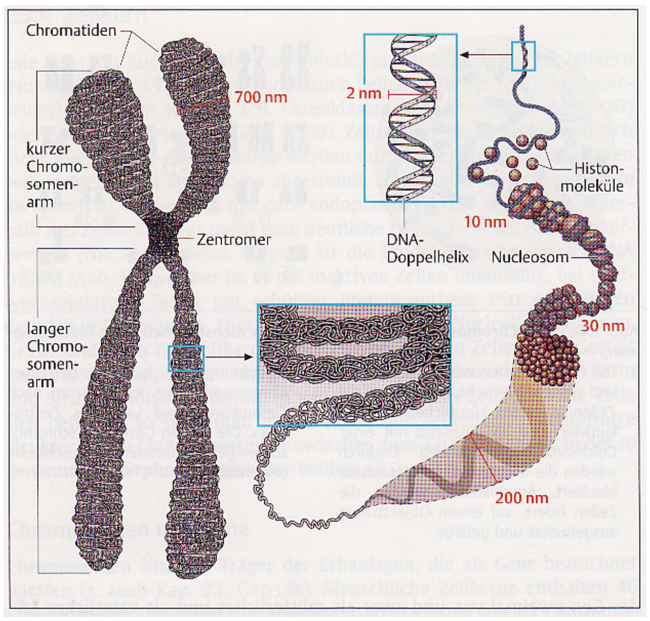
3.1.3 Genetics

In evolutionary algorithms a solution is represented comparable to the genetic information stored in a cell. Therefore the definition of some terms and a short introduction to the evolutionary process will be given here.

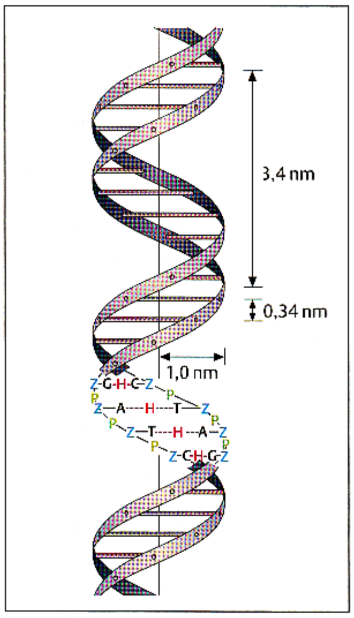
Every *cell* except the red blood cell has at least one *nucleus*. In this nucleus the genetic information is stored in normally more than one *chromosome*, see Fig. 3.2a. The human cell nucleus for instance contains 46 chromosomes in the form of 23 chromosome pairs, see Fig. 3.1. The chromosomes can be distinguished because of their length and shape.

Each chromosome consists of a complex folded and winded double-stranded high polymer in form of a double helix. This molecule is called the **d**eoxyribo**n**ucleic **a**cid or short *DNA* and is constructed of *nucleotides*, see Fig. 3.2b. These nucleotides are the smallest components of genetic information and its order encrypts this information.

The *genotype* embodies the genetic information, i.e., the elementary traits of each life form, while the appearance of the life form is called the *phenotype*. Thus the phenotype describes characteristics like the hair colour or the blood group. The creation of a new genotype, which leads to a new living being but also the whole evolutionary process works in the molecular setting around the DNA and depends on certain factors.



(a)



(b)

Figure 3.2: (a) Scheme of a chromosome; (b) the DNA doulbe helix; from [12]

3.1.4 Evolutionary Factors

Nowadays with the knowledge of genetics the process of evolution according to Darwin can be much more exerted than to his time. As a result modern science describes five evolutionary factors:

Selection Darwin recognised that life forms create much more descendants than necessary to sustain their own kind and that these descendants are not all alike. Moreover he assumed that every life form is challenged all the time for better life conditions, more food and a life partner. He concluded that only the fittest of every kind can survive this every day struggle, what he therefore called *surviving of the fittest*.

This natural selection leads to a gradual improvement of every species and so to a better surviving in its environment. Thus the fitness of a life form can best be measured by the number of the remaining descendants.

Mutation By observing the evolutionary process the alteration of genes from descendants and therefore the growing variety of individuals of the same kind play an important part. The mutation—i.e., the change of the genotype—is in that case the driving force and happens accidentally.

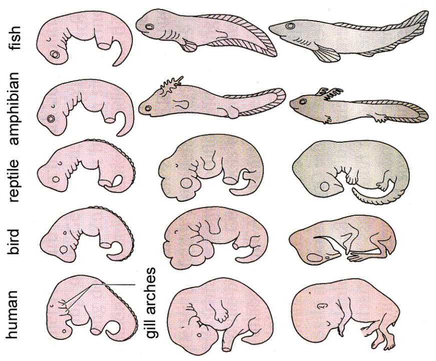


Figure 3.3: Different states of vertebrates in the embryonic development, from [12]

Recombination During the recombination a new gene is created by taking parts of the genes from both parents. This enhances the chance for a better adapted individual but is not a warranty due to the fact that the parts are chosen randomly.

Genetic Drift Accidental changes in the gene pool are called genetic drift. These changes can happen without mutation and selection because of a disease or a natural disaster. This leads to the extinction of one population and another group with a different gene pool will get the chance to spread.

Isolation Groups of individuals can evolve differently when they cannot share the same gene pool anymore. The most common type of isolation is a geographical separation. In that case a population gets divided because of an environmental change and thus each group will develop in a different direction according to its needs.

3.1.5 Proofs

One proof for Darwin's *Theory of Evolution* is that embryos of vertebrates are barely distinguishable. Nearly all vertebrates—including the human kind—for instance reach an embryonic state in which gill arches are applied despite the fact that they are never fully developed, see Fig. 3.3. This can also be seen as a proof that the evolution of vertebrates began with gills breathing life forms.

Rudimental organs represent one of the most impressive proofs, which deal with degenerated organs because they became useless. The human body delivers good examples such as the human tailbone a relic of a tail or the useless muscles of the earlap.

3.2 Evolutionary Process as Metaheuristic

In the late 1950's computer scientists started to experiment with the simulation of basic mechanisms of evolution to solve optimisation problems. Then the concept of the *Evolutionary Algorithm* (EA) was born, see [24]. The idea, as referenced in [24], was to use the computational power to create a bunch of solutions using fast and simple algorithms and selecting the best solution in the end, i.e., population based search, instead of putting all the power in the creation of one solution by a slow and complex approach as it is done normally, i.e., point based search.

Using the known evolutionary factors *recombination*, *mutation* and *selection* new descendants are created based on already existing individuals from a population. A human readable solution itself can be seen as the phenotype while its encoding for the computational use is the genotype—comparable to the chromosome in nature. Therefore it is necessary to find a formal representation—like the DNA and its nucleotides—which allows the EA to apply the *recombination* and *mutation* operators to create descendants in a fast and simple manner. Moreover a function is needed that evaluates the quality of a solution which can be used for the *selection* of the best adapted individuals of a population.

Considering these principles a general EA according to Nissen, see [24], would have the structure as seen in Alg. 1.

One of the most outstanding differences of EAs and the natural evolutionary process is that the replication uses only individuals of the current population. In nature it is mostly common that each individual follows its own reproduction cycle with a—maybe younger or older—partner. A freedom that cannot be given for EAs, because the huge complexity underlying natural reproduction is beyond any current computer system.

The complex natural system needs to be reduced to a manageable structure and therefore in EAs the ascendent population is extinguished and totally replaced by the new population. Thus all individuals in a population are of the same age. This does not mean that all the individuals of the old generation are lost. Commonly some of the best individuals are passed on to the next generation, which means that their chromosome is cloned and added to the current population but the old population itself is lost and their structure cannot be reproduced. This limitation also makes it impossible that an older individual can replicate with an individual created in the current generation.

To control this kind of reproduction a defined number of generations is set at the beginning of every EA, which is usually the breaking condition of the loop in Alg. 1.

Algorithm 1: A general EA scheme

```
output: The best solution found within the EA
begin
    Configure a set of strategy parameters;
    Initialise the population $P(0)$;
    $t \leftarrow 0$;
    Evaluate each individual from $P(0)$;
    repeat                                  // start the reproduction cycle
        $t \leftarrow t + 1$;
        Select;                             // choose from the parents
        Replicate;                          // generate the descendants
        Diversify;                          // vary the descendants
        Evaluate descendants;
        Create new population $P(t)$;
    until breaking condition reached;
    Return best solution found;
end
```

Following the scheme from Alg. 1 there are different approaches to use the idea of evolution for optimisation problems and beyond. Furthermore in [24] Nissen differentiates between four mainstreams of EAs:

- Genetic Algorithm (GA)
- Evolutionary Strategy (ES)
- Genetic Programming (GP)
- Evolutionary Programming (EP)

The implemented EA for this master thesis resembles a *genetic algorithm*. Thus the ideas and functions of the GA will be discussed in more detail. The other three EA mainstreams will be shortly summarised, while more exhaustive introductions can be found in [17, 19, 24, 31].

3.2.1 Genetic Algorithm

GAs were introduced by John Holland in the 1960's, who wanted to explain the mechanisms of adaptive systems by the use of biological evolution. He implemented his ideas in the form of so-called *reproductive plans*. Shortly after, his approach was also used for optimisation problems, see [24].

The main working principle of a GA is the *building block hypothesis*, see [31]. In an optimal solution or phenotype every part of the underlying genotype needs to be arranged in a perfect manner. It is very unlikely that the optimal solution is a member of the initial population but some members might have perfectly arranged parts, like building blocks, in their genotype. Because these individuals are at least partly perfect they will be better evaluated and therefore it is more likely for them to be chosen as parents, following the pattern of nature. As a result the better parts of the parent individuals will be passed on to the descendants and sometimes an individual with more optimal building blocks on the correct positions is created. Repeating this progress often and with a satisfying large population the individuals will develop towards the optimal solution. In technical literature this drift towards better adjusted solutions is often referred to *convergence*.

The GA imitates the evolutionary process to create new populations which consist of individuals. In this abstract case these individuals are now artificial chromosomes. Considering the scheme from Alg. 1 the there introduced methods will be applied in the following way:

Configure

Configuring the strategy parameters includes such decisions as the number of individuals of a population and the number of generations to be created. The most important part of this point is to find a satisfying representation for the chromosome, i.e., the genotype.

For simple problems a string of bits can be used, so that each bit or a set of bits stand for a value in subject to the application. But even in simple cases it proves to be difficult to use a normal binary encoding because of its variable *Hamming distance* and therefore, as discussed in [24], small changes in the string can lead to big changes in the phenotype. A more stable representation would be the *Gray code* which is a binary code with a constant Hamming distance equal one and thus small changes in the genotype remain small in the phenotype.

In recent years more complex chromosome representations were used, which should resemble much to the problem definition and thus appear more natural for the user.

Initialise

To begin with the reproduction cycle a starting population is needed. Normally the individuals of this first population are created using a stochastic approach.

Evaluate

The qualities of all individuals need to be computed such that the population members can be differentiated during the selection process. Therefore a mathematical function is needed to evaluate each individual's quality. This evaluation function is the actual optimisation criterion for the GA, which will be either minimised or maximised according to the problem definition. There are no certain requirements for this function but for a more efficient convergence it is useful that the evaluation function proves to be continuous.

Select

In the selection phase individuals are chosen for the later replication phase, which represent the parents of the new generation. It is necessary to select as many parents as needed to assure that enough individuals can be created so that the population size remains constant.

The ancestors are selected stochastically according to their evaluation. An equally distributed approach would lead to the fact that good and bad individuals would have the same chance to be selected and poor characteristics would be still passed to the descendants, which contradicts to the selection in biological evolution. Considering this makes the separation between evaluation function and a special selection function obvious.

The so-called fitness function fulfils this demand and is linked to the evaluation function. The evaluation function is the measure of the quality of an individual, while the fitness function is the base of the selection process. For each individual is a fitness value computed proportional to its quality. Thus the fitness function assesses the selection likelihood for each individual so that a better solution is more likely to be chosen than a worse. This also follows the *surviving of the fittest* principle from chapter 3.1.4. In [31] it is alluded to the fact that an equivalence of evaluation and fitness would be formally wrong.

Replicate

During the replication new individuals are created based on the parents selected before. This resembles the *recombination* in nature and for GAs it is often called *crossover* because a crossover of each parent's chromosome is done to create a new one. Usually two parents create one or two new individuals during one crossover process. The recombination must be repeated as long as there are enough descendants to build a new population.

While in nature the recombination happens randomly and thus there is no warranty that the new individual will be of better quality, in a GA the recombination can be done following strict rules for a better convergence. These rules improve the chance

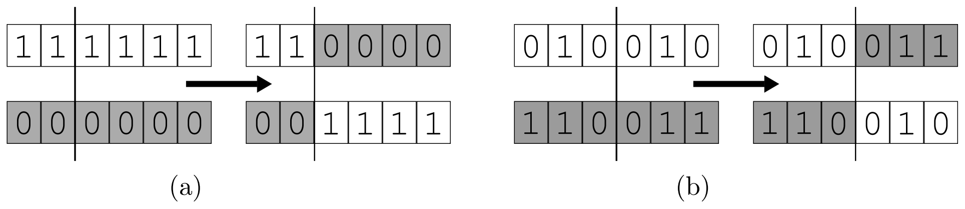


Figure 3.4: Two examples for the one-point crossover

that only the good parts of the parents are passed on to the descendants, which will lead to better solutions during the optimisation process. The whole recombination works on the genotype and therefore all crossover operations are done with the chromosomes. The following recombination types will help to understand the proceeding and were used for this diploma thesis. While the concrete implementation of these operators according to the defined problem will be discussed later, an overview of the operators will be given here.

One-point crossover Introduced by Holland [10] himself the one-point crossover is a simple working crossover and a perfect example of the building block hypothesis, see 3.2.1. Here two parents build the base for two new individuals. The idea is that a randomly selected point splits the parents' chromosomes into two building blocks. These blocks are passed on to the descendants, one block from every parent. Thus each new individual inherits the first part from one parent and the second part from the other. This also means that identical individuals create descendants indistinguishable from their parents. As seen in Fig. 3.4a the one-point crossover can produce descendants that vary in many ways from their parents. On the other side there will be no differences in a bit where both parents share the same value, as shown in Fig. 3.4b.

Using two splitting points will help to find better building blocks, which is realised in the *two-point crossover*, as the name indicates. Furthermore a variable number of splitting points leads to the *n-point crossover*.

Position sorting crossover The previously described crossover, in all its variations, takes for each part the order information from only one parent into account. Thus, like many other crossover operators, the order common to both parents is not preserved. An approach which considers both orders is featured in [15] called the position sorting crossover. Here the average position for each value is computed according to both of the parents' locations. Sorting the values according to the calculated positions results in the new sequence, which represents the new descendant. Therefore the position sorting crossover can only create one child from two parents. See Fig. 3.5 for an example: the relative

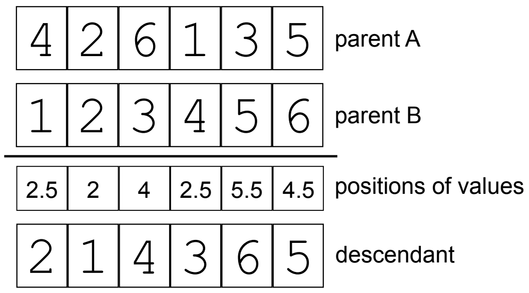


Figure 3.5: An example for the position sorting crossover

position of elements x and y are preserved in the newly created child. For a better understanding an example of this crossover is given in Fig. 3.5. There it is easy to see that one value, which stands in both parents before another, is also positioned before it in the descendant.

Edge recombination crossover In [37] an operator is described especially designed for the, in [20] introduced, *travelling salesman problem* (TSP)—the edge recombination crossover.

Usually chromosomes, encoding a TSP route in a graph, are represented as a string of vertices and two neighbouring vertices in the string are treated as an edge. Therefore a route like $[a, b, c, d, a]$ stands for the edges ab, bc, cd, da. When dealing with symmetric problems of that kind the direction of the edges does not matter and thus the route $[a, d, c, b, a]$ would encode the same solution in the phenotype but with complete different representations in the genotype.

The edge recombination crossover solves this problem because it does not operate on the chromosome itself but extracts the edges from the string to create a so-called *edge list*. This edge list contains for each vertex a list of its neighbours from both parent strings. Thus each list linked to a vertex holds from two, both parent neighbours are identical, to four, all neighbours are different, elements. If a vertex in the edge list is linked to less vertices it is more likely that the represented edges are part of a good or maybe optimal solution and therefore these edges should be selected for the new individual.

To follow the procedure of the edge recombination crossover an example for an edge listis given in Fig. 3.6. The first step is to build the edge list. The initial vertex *current* can be chosen arbitrarily and is then deleted from the lists linked to each vertex. If there are any vertices linked to *current*, i.e., the list of *current* is not empty, then the vertex with the least elements in its list is selected as the next *current* and saved as the next element in the descendant

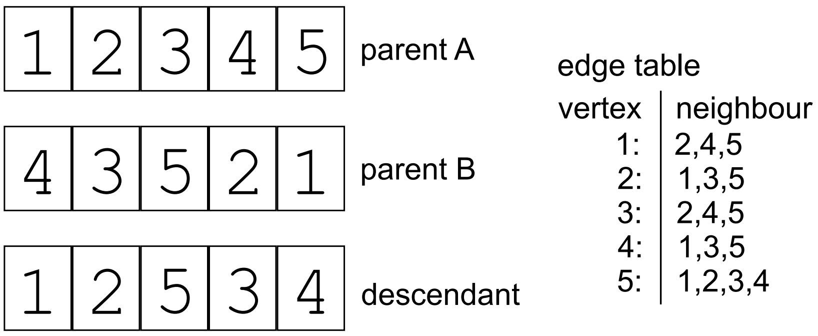


Figure 3.6: An example for the edge recombination

string. Ties are broken randomly. This process is repeated until all vertices are members of the descendant. Although this process can be extended to more than two parents, like in [33], usually the offspring is created from only two parents, see [37].

Diversify

In nature errors occur during the recombination process. In this diversification—also called mutation—Darwin saw the driving force for the variety of individuals. In contrast to recombination the mutation in a GA happens accidentally. Moreover not every individual will be mutated. Thus a probability factor is needed, which defines how many members of a population will be affected by mutation. This *mutation rate* is one more important strategy parameter which must be defined in the configuration phase. Due to the fact that the mutation alters parts of the chromosome, it does not create a new chromosome but replaces the original one. The primary goal of mutation is to overcome local optima during the optimisation process. While in other EA variants the mutation plays a leading part, in GAs it is only of subordinate importance. To the most common approaches belong these ideas:

One-bit inversion A simple approach to alter the genotype is to change the value of one of the chromosome's bits. This is attained by selecting a random position and inverting the bit value there. An example of this process can be seen in Fig. 3.7a. The change is affecting the genotype only a little and choosing a proper representation the effect on the phenotype will be small, too. Moreover there is to note that this mutation type just alters one bit of the chromosome but does not effect any other value or their arrangement.

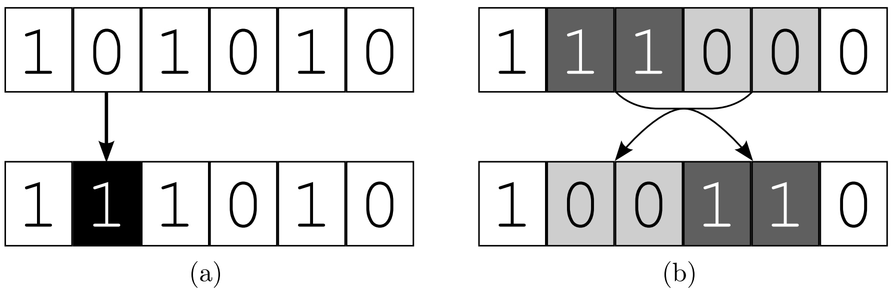


Figure 3.7: (a) Shows an example for the one-bit inversion; (b) for the two-bit switch

Two-bit switch Using more complex chromosome representations simple operations like the one-bit inversion would lead to undefined solutions. Then another approach of mutation is needed like swapping the position of two values. By selecting two random positions and interchanging the values there, the mutated individual still features the same values but a rearranged order, as seen in Fig. 3.7b.

Create new Population

As already mentioned the old population is at the end of one cycle extinguished and replaced with the currently new created generation. This process is called *generational replacement*. It might happen that none of the new individuals is better than their ancestors, which would mean that there was no improvement during this generation and this could lead to even worse results later on. To minimise this risk it appears useful to let some of the best ancestors survive.

This idea of *elitism* counteracts the discussed problems of generational replacement by keeping some of the best individuals from the old generation. Therefore it is either possible to create less individuals during the recombination or to skip the worst individuals by keeping the best ancestors. Sometimes elitism can lead to a too fast convergence. *Weak elitism* deals with this problem in the way that the ancestors are passed to the next generation only in a mutated form.

3.2.2 Evolutionary Strategy

While GAs are best used for combinatorial problems the idea of an ES is to optimise functions defined on real numbers. ESs were introduced in the 1960s by Ingo Rechenberg and Hans-Paul Schwefel for optimisation in the technical-physical field, see [31].

In the case of ES the chromosome is represented by a string of real numbers whereas every number resembles an optimisation factor. In an ES the current population usually is called with the Greek letter μ and the set of descendants with λ.

The process of an ES resembles the already pictured general EA scheme shown in Alg. 1. First the starting population is created stochastically. To get the next population new individuals are created on the base of the current population. The relation between the sizes of μ and λ is usually set from 1 to 7, i.e., every individual has one to seven descendants.

The idea is to create a vast number of new individuals therefore the selection process can deal with a lot of opportunities to choose. This is possible because of very fast recombination and mutation algorithms, while the mutation proves to be the more important operator.

For recombination two different crossovers are commonly used, see [24]. The *discrete recombination* decides randomly for each position in the string if the value from the first parent or the second is taken and passed to the descendant string. Another approach—*the intermediate recombination*—is to compute for every number in the string the mean from both parents' numbers and thus create the values for the new individual.

During the mutation for each real number from the string a random value is generated and added to the number. This is achieved by using a normally distributed function with an expectation equal to zero, i.e., it is, very likely that the real numbers are changed only slightly or not at all.

In the selection phase μ individuals have to be chosen to build the next population. In ESs *elitism* is always applied, which means that the best individuals of both sets μ and λ will be taken. One normally differentiates between two selection strategies:

Plus strategy There the best μ chromosomes are chosen from both the ancestors and the descendants. This means that the next generation includes individuals from $\mu + \lambda$ and therefore this kind of ES is also called $(\mu + \lambda)$-ES.

Comma strategy The other approach is to create the new population on the base of the best individuals from λ alone. Because in that strategy the ancestors are not considered in the first place, a check is needed if the best of all individuals is a member of the older population. If so then this individual is added to the next population. This approach leads to the so-called (μ, λ)-ES.

3.2.3 Genetic Programming

While normally the length of the chromosomes are fixed, some problems require a variable chromosome size. GPs use chromosomes with a totally different structure, which can also be interpreted as programs. Therefore GPs create programs—or more precisely program codes—that optimise their I/O behaviour according to the problem definition. John R. Koza was the main driving force in the development of GPs [24].

Thus for a GP a finite set of functions and function symbols $\mathcal{F}$ and a finite set of tokens $\mathcal{T}$ (constants and variables) must be defined. Each chromosome $\mathcal{C} = \mathcal{F} \cup \mathcal{T}$ is therefore a compound function and represented as a list of these components.

For turning this formal list into source code that can be compiled and run, a programming language is needed which is comparable to this chromosome structure. In principle any programming language would fulfil this task but, because of its list representation, LISP (*List-Processing*) provides the perfect base for this requirement.

The structure of a GP is the same as in Alg. 1 with the exception that there is no mutation. As any EA a GP also begins with a starting population which is created stochastically whereas attention should be paid that the individuals do not get too complex.

Computing the fitness, i.e., the fitness of a program, rather poses a challenge. Therefore the fitness is assessed based on some case studies and the quality of their program outputs. The selection process afterwards is comparable to that of a GA.

The recombination can be done by choosing a logically correct part of each parent chromosome and interchanging these parts for the descendant chromosomes. Again attention should be paid that the individuals will not become too complex.

3.2.4 Evolutionary Programming

EPs are only a small fragment in the world of EAs but non the less interesting. They were originally introduced by Lawrence J. Fogel, Alvin J. Owens and Michael J. Walsh to generate intelligent Automata, while EPs were later also adapted to deal with optimisation, see [24]. Normally the purpose of an EP is to optimise a multidimensional function with the global optimum in its origin.

The interesting aspect of EPs is that the evolutionary process is done from the point of view of the phenotype. The idea is to mutate the individual itself and not the chromosome to reach the optimum and therefore the mutation is the only operator needed.

Despite the missing recombination the process of the algorithm is comparable to an ES. First the starting population is generated stochastically. Then a copy of the population is created and the mutation is applied to all the copied individuals. After that the population has to be reduced to the original size using the selection operator.

The selection is not directly done through the fitness instead each individual i of the big population must compete against a set of other individuals from the same population. The contest is then achieved by computing the fitness of the individuals and comparing the fitness of i to the fitness of its competitors. Now counting every time i wins, will lead to a ranking of all the individuals. The best of this ranking are chosen as the base for the next generation, which automatically guarantees that the very best individual is part of the survivors.

3.3 Other Metaheuristics

The algorithm implemented for this master thesis resembles a GA. But beside the various kinds of EAs the field of metaheuristics offers many different other approaches. Some of these have already been pursued for solving RCCSTD, namely the *ant colony optimisation* (ACO) and the *variable neighbourhood search* (VNS). Because the results of the ACO will be compared with the GA in Sec. 6 and the VNS was used to improve the GA results the ideas of both metaheuristics will be shortly described here.

3.3.1 Ant Colony Optimisation

Another bionic approach is the ACO, which belongs to the field of swarm intelligence algorithms, see [32]. Due to the fact that simple ants are able to solve complex tasks such as food transportation and finding the shortest path, agent ants are created in the computer to simulate the cooperative behaviour of real ants.

In the real world ants orientate and coordinate themselves along so-called *pheromone* trails. Pheromone is an olfactive and volatile secretion that ants atomise along their way. On the other hand ants follow the smell along a pheromone trail and the larger the amount of this secretion, the larger is the probability that the ants will follow that path, while the pheromone evaporates over the time.

Therefore if an ant trail is blocked by a barricade, then some ants will try to bypass along the one side and others along the other side. The ants that choose the shorter way will be faster and thus more ants will pass this section, which increases the

Algorithm 2: A general ACO scheme

```
output: The best solution found within the ACO
1  begin
2      Initialise the pheromone trail;
3      repeat
4          for each ant do
5              Construct solution using the pheromone trail;
6              Update the pheromone trails;
7          end
8      until breaking condition reached ;
9      Return best solution found;
10 end
```

pheromone intensity along this path and therefore more and more ants will choose this way around the barricade. Because of this mechanism ants are able to find the shortest path between two points.

A possible computer simulation of this process, according to [32], can be seen in Alg. 2. After the pheromone trail is initialised the optimisation process starts. Within this process solutions are constructed (line 5) according to a simple stochastic greedy method that creates a solution for each ant. Then the pheromone must be updated in relation to the generated solutions. To avoid a too fast convergence each pheromone value is reduced automatically by a fixed proportion, which simulates the pheromone evaporation in nature. In the end, after a certain time period, the best solution found will be returned.

3.3.2 Variable Neighbourhood Search

A VNS [16] in general is a metaheuristic that consists of a local search procedure and is extended by a method to overcome local optima, by exploring predefined neighbourhoods for better solutions. This is achieved either randomly or a set of neighbourhoods is fathomed systematically. Thus the idea of VNS is to find different local optima and that the global optima is a local optima to the other neighbourhoods.

The deterministic version of VNS is the so-called *variable neighbourhood descent* (VND). The VND builds the base for VNS by exploring the neighbourhoods in descent to a local optima. Thus the VND relies on the definition of proper neighbourhood structures N_l, with $l = 1 \ldots l_{max}$. Starting with $N_1(x)$ and x, as the initial solution, an improvement is searched within the current neighbourhood, in that case

N_1. If this is not possible the next structure N_{l+1} in the hierarchy is taken and observed. If an improvement was found the algorithm returns to the first structure and restarts the search considering the new local optima x.

For VNS, which is a stochastic algorithm, a set of neighbourhood structures N_k, with $k = 1 \ldots k_{max}$, is defined at first. Then each iteration of VNS contains three steps:

shaking randomly takes an initial solution x from the current neighbourhood N_k.

local search generates a new solution x'. If x' is better than x then $x = x'$ and the procedure is restarted with N_1.

move If no improvement was found, the algorithm moves to the next neighbourhood structure N_{k+1}.

Instead of local search a VND method is often applied for VNS. Therefore the neighbourhoods can be explored systematically by the deterministic VND, while the VNS sets the starting point randomly.

3.3.3 Hybrid Metaheuristics

The use of hybrid metaheuristics became more and more important over the last years. For many optimisation, either real-life or classical, problems the best results were found by this approach, see [32]. In [32] four different possible combinations for metaheuristics are mentioned:

- combining some metaheuristics with each other
- combining metaheuristics with exact methods
- combining metaheuristics with constraint programming
- combining metaheuristics with machine learning

Within this master thesis the developed GA is combined with the VNS from [28], which therefore resembles the first item in the list mentioned above. Therefore the approach to combine two metaheuristics will be discussed in the following. A combination of GA with another heuristic is also called a *memetic algorithm*. Good results combining a GA with VND were achieved in [25].

Moreover a graduation [32] between a *low-level hybridisation* and a *high-level hybridisation* can be made. In the low-level approach a given function of one metaheuristic is solved by another metaheuristic, thus the algorithms are linked to each other. But this type of hybridisation is barely used.

In the high-level approach independent metaheuristics are used like a pipeline each using the output of the previous algorithm as its input. Thus the metaheuristics are executed in a sequence.

A popular combination scheme is to create the initial population by greedy heuristics, which is also done for this master thesis. Another approach is to use a metaheuristic to improve the already generated solutions. Within this work this is done twice with the help of the VNS. Once to improve the best individuals every few generations and on the other hand to improve the very best individual found at the end of the reproduction cycle.

CHAPTER 4

A Memetic Algorithm for the RCCSTD Problem

Based on the mechanisms described in Sec. 3.2.1 a genetic algorithm (GA) was designed for this master thesis to reconstruct cross-cut shredded documents as defined in Sec. 2.1. In this section the main ideas and the used operators will be described.

Previously other reconstruction approaches have been implemented for this framework like a *variable neighbourhood search* (VNS) described in [23] and moreover an *ant colony optimisation* (ACO), *based on integer linear programming techniques* (ILP) and some *construction heuristics*, see [28].

The VNS was also used to further improve the solutions created by the GA. This approach, to combine a GA with another metaheuristic, is also called a *memetic algorithm*. Due to the fact that for this thesis only the GA part was newly created, the described algorithm will always be referred to as GA and not as memetic algorithm.

Taking the classic structure of an evolutionary algorithm, like shown in Alg. 1, the procedure was adapted for the specific needs of this work. The new structure for this thesis can be seen in Alg. 3.

The main difference to a classic GA as shown in Alg. 1 is the fact that the evaluation of individuals is done directly after the recombination or mutation and thus its value is computed by the operators. This is done because in certain test situations only the better individual was chosen from recombination operators that created two descendants. Therefore the individuals must be evaluated at once. In addition to the GA from Alg. 1, a local search procedure, the VNS, is performed.

The reproduction cycle is repeated until the a certain number of generations is reached. This specific number is assigned as an input variable. It is also the only breaking condition for the loop, because preliminary tests have shown that even after long times of stagnation an improvement was still possible. Another input variable is the population size, which defines the number of individuals for each population $P(t)$ at the moment t.

The initialisation of the population (line 3) is performed using two construction heuristics, namely the *row building heuristic* and the *Prim-based heuristic*, originally introduced in [28]:

row building heuristic takes into account that usually for each line of a text document the beginning and the end of the line is white. Thus this heuristic takes a shred with a white side on the left and repeatedly adds shreds to this line, using best fit, until a shred with a white right side is found and therefore the end of line reached. Then a new line is started following the same procedure.

Prim-based heuristic is based on the algorithm of Prim [30], which was designed to find a minimum spanding tree. Like the Prim algorithm this heuristic starts with an arbitrarily chosen shred, in that case at $p = (1, 1)$. The next shred is added arround the already existing solution and the new shred must stick to the solution on at least one side. To find the next shred a best fit approach is used.

The rest of this section is dedicated to the methods used in the main loop and the operators created for them will be discussed.

4.1 Selection

For selecting the parents individuals are randomly chosen from the preceding population using an equally distributed function. This seemingly simple approach takes into account that the *error estimation function* (EEF) from Sec. 2.2 performs well for this assignment. Tests have shown, however, that for many instances solutions can be created whose objective with respect to EEF is smaller than the original document evaluation. Therefore, when relying on a fitness function, which is highly dependent of EEF, the probability that correct solution features are preferred, is reduced.

To compensate this simple equally distributed approach only the better offspring is taken from crossover operators which create two descendants. This leads to the assumption that mostly better individuals are created for the next generation. Moreover elitism is performed in a way that ten percent of the best individuals from the

Algorithm 3: The GA for this thesis

input : Number of generations *generations*, population size *size*
output: The best solution found within the GA

```
1  begin
2      $t \leftarrow 0$;
3      Initialise the population $P(t)$;
4      repeat                                      // start the reproduction cycle
5          $t \leftarrow t + 1$;
6          Select($P(t-1)$);                           // choose from the parents
7          Recombine($P(t)$);               // generate & evaluate the descendants
8          Mutate($P(t)$);                                // mutate descendants
9          Create new population $P(t)$;     // new population from descendants
10         ImproveSolutions($P(t)$);                 // improve some individuals
11     until $t == generations$;
12     ImproveFinalSolution($P(generations)$);       // improve best individual
13     return best individual;
14 end
```

previous population are passed unchanged to the next. This approach was also verified by preliminary tests.

4.2 Recombination

For the recombination several crossover operators have been designed. Based on the crossovers mentioned in Sec. 3.2.1 these operators have been adapted according to the specific problem of document reconstruction.

Algorithm 4: Horizontal block crossover

```
input  : $solution1$, $solution2$: the parent individuals
output: $newSolution1$, $newSolution2$: the descendant individuals
begin
    Create $newSolution1$, $newSolution2$;
    Compute $splitHorizontal$;                                  // randomly chosen
    for $i \leftarrow 0$; $i < splitHorizontal$; $i{+}{+}$ do
        $row$ = $solution1$.getLine($i$);                 // copy top half from $solution1$
        $newSolution1$.add(row);
    end
    for $i \leftarrow splitHorizontal$; $i < number\ of\ lines2$; $i{+}{+}$ do
        for $j \leftarrow 0$; $j < end\ of\ line$; $j{+}{+}$ do
            $shred$ = $solution2$.getLine($i$).getColumn($j$);
            if $shred \notin newSolution1$ or $shred == virtualShred$ then
                $row$.add($shred$);                        // bottom half from $solution2$
            end
            $newSolution1$.add($row$);
        end
    forall $shred \notin newSolution1$ do                      // copy remaining shreds
        add $shred$ to empty space in $newSolution1$ using BestFitHeuristic;
    end
    Repeat lines 4 to 18 for $newSolution2$;
    return $newSolution1$, $newSolution2$;
end
```

4.2.1 Horizontal block crossover

The *horizontal block crossover* (HBX) follows the idea of the *one-point crossover* (1PX). Nevertheless the basic approach described earlier cannot be directly adopted. Due to the more complex representation of RCCSTD each shred has to occur exactly once. The scheme of HBX is printed in Alg. 4.

While for 1PX a splitting point is chosen, which defines where the parents will be taken apart, for this two-dimensional problem a line as breakpoint is needed. Moreover, this line is chosen randomly by simulating a Gaussian distribution function and of course according to the shorter of the two parents. For simplicity two equally distributed random integers were generated and added, which resembles the throw of two dices. The Gaussian distribution was chosen, because bigger blocks can be created, when it is more likely that the two parts split apart are of equal size.

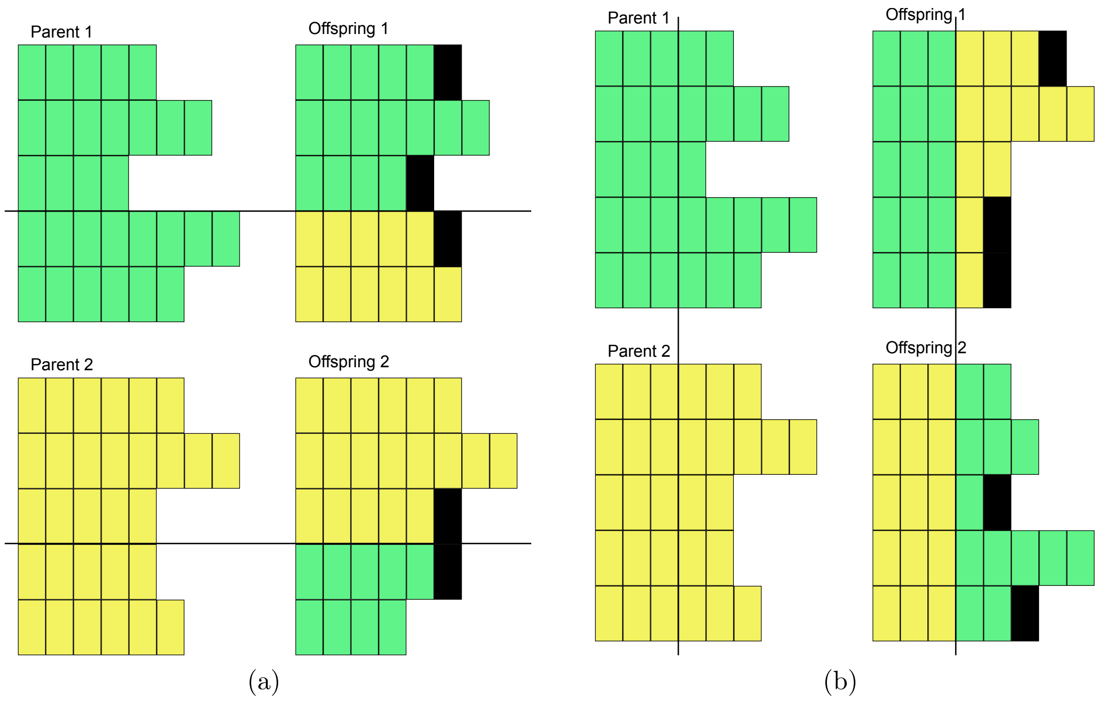


Figure 4.1: (a) Shows an example for HBX; (b) shows an example for VBX; in both figures the black shreds are added using best fit

For `newSolution1` the upper part of `solution1` and the lower part of `solution2` is taken, while for `newSolution2` it is the other way round. The lines 4 to 18 in Alg. 4 describe how to create `newSolution1`. The whole procedure must be repeated again for `newSolution2` as hinted in line 19. An example for HBX is presented in Fig. 4.1a.

In line 4 to 7 the upper part is inherited unchanged from one parent like for the 1PX. It can, however, happen that one shred occurring in the upper part of one parent also occurs in the lower part of the second parent. If the lower part would just be copied, some shreds could be found twice in the descendant while others would be left out.

Therefore a check, like the one in line 11, is needed to make sure that only new shreds will be chosen for the lower part. Some shreds might be left out during this procedure.

These are added to the end of each line or instead of an empty shred, the so-called virtual shred, using a best fit heuristic. This heuristic computes the EEF value for every possible position and adds the shred at the position with the lowest EEF value.

Algorithm 5: Vertical block crossover

```
input  : $solution1$, $solution2$: the parent individuals
output: $newSolution1$, $newSolution2$: the descendant individuals
begin
    Create $newSolution1$, $newSolution2$;
    Compute $splitVertical$;                                // randomly chosen
    for $i \leftarrow 0$; $i < number\ of\ lines1$; $i{+}{+}$ do
        for $j \leftarrow 0$; $j < splitVertical$; $j{+}{+}$ do
            $shred$ = $solution1$.getLine($i$).getColumn($j$);
            $row$.add($shred$);                    // copy right half from $solution1$
        end
        $newSolution1$.add($row$);
    end
    for $i \leftarrow 0$; $i < number\ of\ lines1$; $i{+}{+}$ do
        for $j \leftarrow splitVertical$; $j < end\ of\ line2$; $j{+}{+}$ do
            $shred$ = $solution2$.getLine($i$).getColumn($j$);
            if $shred \notin newSolution1$ or $shred == virtualShred$ then
                $newSolution1$.getLine($i$).add($shred$); // left half from $solution2$
        end
    end
    forall $shred \notin newSolution1$ do                // copy remaining shreds
        add $shred$ to empty space in $newSolution1$ using BestFitHeuristic;
    end
    Repeat procedure for $newSolution2$;
    return $newSolution1$, $newSolution2$;
end
```

While the copying of the first part can be done in $O(n)$, in the worst case the upper part of one parent is equal to the lower part of the other. Thus the best fit heuristic must be used for half of the shreds, which leads to a worst case complexity of $O(n^2)$ for HBX.

4.2.2 Vertical block crossover

The *vertical block crossover* (VBX), shown in Alg. 5, can be seen as the complimentary approach of HBX. In that case the splitting line is moving vertically through the `solution` and cuts it into two halfs. While HBX passes long horizontal parts on to the descendants, the vertical block crossover was designed to hand long vertical parts on to the offspring.

Taking the longest column of both parents into account, the line is again computed with the pseudo Gaussian distribution used in HBX. Both descendants inherit the left part unchanged from one parent. For the right part it is again important to make sure that in the end each shred occurs only once. Remaining shreds are analog to HBX added using a best fit heuristic, see the example in Fig. 4.1b.

Because of the best fit process the complexity of VBX is, like that of HBX, $O(n^2)$.

4.2.3 Best neighbour crossover

As already mentioned above, it might occur during the optimisation that there are empty slots within the `solution`. To handle such slots efficiently they are filled with placeholders—the so-called virtual shred. Due to the general design of the operators utilised it is, however, likely that empty shreds are introduced. Obviously, this leads in most cases to `solutions` with large empty regions.

To overcome this drawback, i.e., to delete obsolete virtual shreds to make `solutions` more compact, the *best neighbour crossover* (BNX) was designed, which scheme can be seen in Alg. 6.

The idea is to take two individuals and decide for each position of the offspring whether to take the shred from one parent or the other. This means that in certain situations if one block from the first parent and the other block from the second are positioned next to each other in the parents and fit together well than they will be passed to the offspring.

Due to the fact that both parents will be of different form, one descendant will inherit the shape of the first parent and the other the shape of the second parent. Beacuse the virtual shreds are ignored the offspring will be more compact in shape than the parents'.

For `newSolution1` the first non virtual shred from `solution1` is taken and added, see line 3. Following the structure of `solution1` all the other shreds are added by iteratively comparing if the shred from `solution1` or `solution2` fits better, according to EEF, at the current position, which is done in the lines 11 to 15.

If one of the two possible shreds is a virtual shred, than the other snippet will be taken. On the other hand if both are virtual shreds than they are skipped and the shreds on the next position in the parents' line will be taken into account, compare lines 8 and 9. In `newSolution1` the current position will of course not be skipped

Algorithm 6: Best neighbour crossover

```
input  : $solution1$, $solution2$: the parent individuals
output: $newSolution1$, $newSolution2$: the descendant individuals
begin
    Create $newSolution1$, $newSolution2$;
    Copy upper left non virtualShred from $solution1$ to $newSolution1$;
    for $i \leftarrow 0$; $i < number\ of\ lines1$; $i{+}{+}$ do
        for $j \leftarrow 1$; $j < number\ of\ columns1$; $j{+}{+}$ do
            $shred1 = solution1$.getLine($i$).getColumn($j$);
            $shred2 = solution2$.getLine($i$).getColumn($j$);
            if shred1 and/or shred2 is virtualShred then
                ignore the virtualShred and add the other or none
            else
                if $shred1$ fits better than $shred2$ at current position then
                    $newSolution1$.getLine($i$).add($shred1$);
                else
                    $newSolution1$.getLine($i$).add($shred2$);
                end
            end
        end
    end
    Repeat procedure for $newSolution2$;
    return $newSolution1$, $newSolution2$;
end
```

and thus the offspring will not contain any virtual shreds in the end, making the descendants more compact.

The procedure from lines 3 to 18 must be repeated for `newSolution2` starting with the first non virtual shred of `solution2` and inheriting the structure of the latter parent.

To create the offspring both parents are iteratively observed but each position is only visited once for each descendant, which leads to a runtime of $O(2 \cdot n)$ for one child, with each individual containing n shreds. Thus the whole crossover has a complexity of $O(n)$.

Algorithm 7: 2D position sorting crossover

```
input  : $solution1$, $solution2$: the parent individuals
output: $newSolution1$, $newSolution2$: the descendant individuals
1  begin
2      Create $newSolution1$, $newSolution2$;
3      forall $shreds \in solution1$ do
4          Get $position1$ of $shred$ in $solution1$;
5          Get $position2$ of $shred$ in $solution2$;
6          $relativePositionX[shred] \leftarrow \lfloor (position1.x + position2.x)/2 \rfloor$;
7          $relativePositionY[shred] \leftarrow \lfloor (position1.y + position2.y)/2 \rfloor$;
8      end
9      forall $shreds \in solution1$ do
10         Add $shred$ to $newSolution1$ in line $relativePositionY[shred]$ using
           BestFitHeuristik for the column;
11     end
12     forall $shreds \in solution1$ do
13         Add $shred$ to $newSolution2$ in column $relativePositionX[shred]$ using
           BestFitHeuristik for the line;
14     end
15     return $newSolution1$, $newSolution2$;
16 end
```

4.2.4 2D position sorting crossover

The *2D position sorting crossover* (2PSX), which is based on the idea of the *position sorting crossover* introduced in [15], picks up the idea to position a shred within the offspring in relation to the position of the shred in both parents.

For this purpose an intermediate position for each shred is computed, which is then used as a decision basis for placing each shred in the offspring. Alg. 7 outlines the basic principle of 2PSX in pseudo-code.

Since it is most likely that multiple shreds are assigned to the same intermediate position, only the exact computed x coordinate is used. The second coordinate is determined using the best fit heuristic.

A second offspring is generated analog by using the exact y coodinate and best fitting the x coodinate. Because of the best fit heuristic the complexity for this crossover is $O(n^2)$.

4.2.5 2D EDGE RECOMBINATION

The *2D edge recombination* (2DERX), see Alg. 8, is modelled on the *edge recombination crossover* (ERX) from [37]. The original ERX was considered for the one dimensional symmetric travelling salesman problem, where two parents are creating one descendant. Thus each list linked to a vertex holds from two, both parent neighbours are identical, to four, all neighbours are different, elements.

In the case of this master thesis the situation is different because cross-cut document reconstruction is two dimensional and the relation between two shreds is not symmetric. This means that usually there is a difference if one shred is left or right of another snippet and likewise for the top and bottom relations.

Thus there is not only one edge list but four *neighbour lists*, in each case one for the relation of the top, the bottom, the left and the right position next to each shred. Therefore each list could only contain two entries for a two parent approach. Preliminary tests have shown that with this approach the descendant is only a copy of one of the parents.

Therefore in a first test phase the number of parents was varied between two and six. A similar approach with more than two parents was already followed for the classic ERX operator in [33]. In the end the number of parents was increased to four for this operator. By this enhancement the number of list entries is one (in all `solutions` one shred has the same neighbour) up to four (all neighbours are different for a shred in each parent), which rather resembles ERX.

Every new shred that is added to `newSolution`, is then chosen according to these neighbour lists. At the beginning these lists have to be initialised, which is done in the lines 3 to 5 of Alg. 8. Moreover is a visualisation of these lists shown in Fig. 4.2.

A starting shred is chosen based on the number of entries in all four lists together and added as the first shred to the newly generated offspring. After this shred is added, it must not be considered as a possible neighbour any more and thus it is deleted from the neighbour lists $left$, *right*, *top* and *bottom* of all shreds.

Furthermore it is necessary to know which positions are empty and which are already filled with a shred. It would be too time-consuming to search the whole structure of `newSolution`, to find a free place for the next shred. Therefore in line 8 the *currentFreePositionsList* is introduced. Each time a shred is added, a check is made whether there is an empty or filled neighbour at each side of the new shred. The empty spaces are added to the *currentFreePositionsList*, which therefore organises the possible locations for the next shreds.

Algorithm 8: 2D edge recombination

input : *solution*1, *solution*2,*solution*3, *solution*4: the parent individuals
output: *newSolution*: the descendant individual

```
begin
    Create $newSolution$;
    forall $shred \in solution1$ do
        Create the neighbourhood lists $left$, $right$, $top$, $bottom$ for each side of
          $shred$ taking $solution1$, $solution2$,$solution3$, $solution4$ into account;
    end
    Get starting $shred$ and add to $newSolution$;
    Delete $shred$ from $left$, $right$, $top$, $bottom$;
    Add free spaces to $currentFreePositionsList$;  // the spaces around shred
    $counter = 1$;                                  // the first shred was already added
    while $counter < numOfShreds$ do
        if $currentFreePositionsList.isEmpty$ then  // no free positions left
            Get new starting $shred$ and add to $newSolution$;
            Delete $shred$ from $left$, $right$, $top$, $bottom$;
            Add new free spaces to $currentFreePositionsList$;
        else                                        // there are free positions left
            Choose next $position$ to be filled from $currentFreePositionsList$;
            Choose $shred$ according to $left$, $right$, $top$, $bottom$;
            Add $shred$ at $position$ to $newSolution$;
            Delete $shred$ from $left$, $right$, $top$, $bottom$;
            Delete $position$ from $currentFreePositionsList$;
            Add new free spaces to $currentFreePositionsList$;
        end
        $counter$++;
    end
    return $newSolution$;
end
```

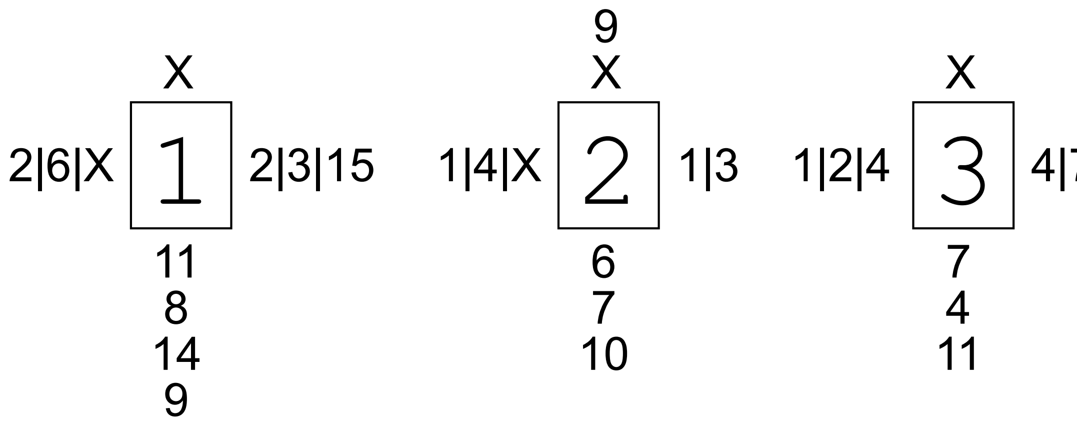


Figure 4.2: An example of neighbour lists of 2DERX

After the first shred is added, there are possible neighbours stored in the neighbour lists and free spaces available to add the remaining shreds. This is then done in the loop from line 10 to 24. The more interesting part are the lines 16 to 21. At first, line 16, the position is chosen, where the next shred will be added. This is decided according to the number of possible shreds that are available at each position and the position with the least opportunities is taken, because this means that some parents match in that specific case.

There exist one, the position is on the brink, up to four, the free position is surrounded with shreds, possible neighbours, depending on the position of the free space. Therefore the neighbour lists of the surrounding shreds must be taken into account to select the next shred, as is done in line 17. If there is already a shred located above the free position the *bottom* list of this shred is taken and in the same way the other neighbour lists are handled. If one or more shreds occur in each of the considered lists, this one shred is chosen or in the case of more opportunities it is decided randomly. Otherwise if the lists have no shred in common, all the shreds from the lists are considered and one of them is chosen randomly. This new shred is then added to `newSolution`.

Then the shred is deleted from the neighbour lists, line 19, and the now filled position from the *currentFreePositionsList*, line 20. Afterwards the free positions around the new shred are added to the *currentFreePositionsList*, line 21. If a position would be empty but in the neighbour lists surrounding the space are no entries, this position is obviously not added, because then no shred could be found to fill the space.

What to do, if there are no free positions left, which can happen, if all the neighbour

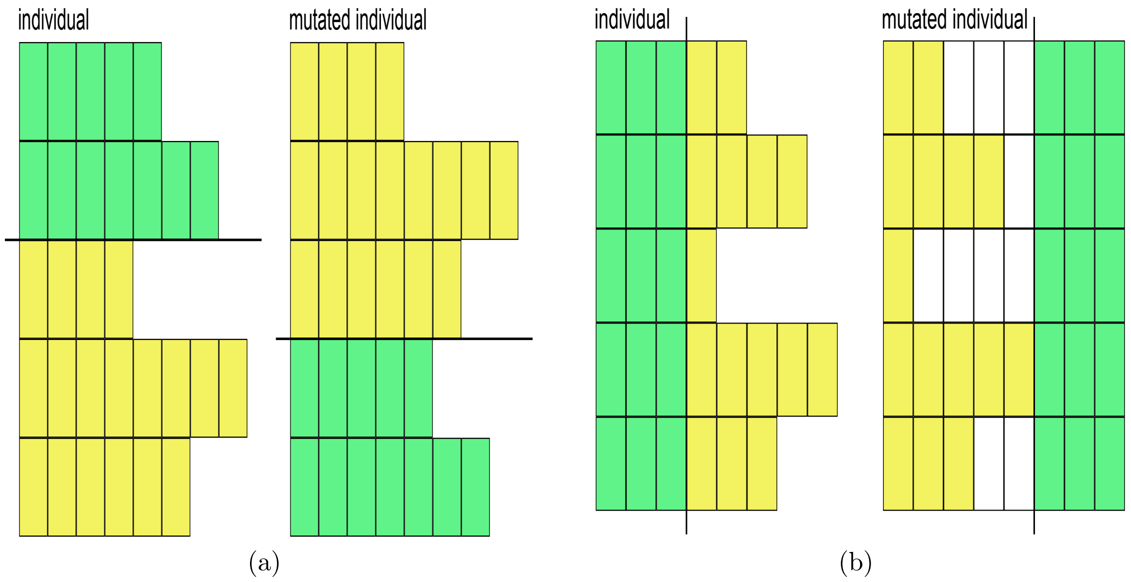


Figure 4.3: (a) An example for a horizontal flop; (b) an example for a vertical flop

lists of the already added shreds are empty, is shown in the lines 12 to 14. This happens to be a kind of restart and the same is done as in line 6 to 8. A new shred is chosen, according to the number of possible neighbours and added to `newSolution` being the first shred in a new line. Therefore new positions can be added to the *currentFreePositionsList* and the process from line 16 to 21 starts again. The whole loop is repeated until all shreds are added to `newSolution`.

The updating of the neighbour lists already has a complexity of $O(n^2)$, for n shreds. The other operations can be achieved in linear time, which leads to a complexity of $O(n^2)$ for 2DERX.

4.3 Mutation

In addition to the main purpose of these operators, i.e., to mutate individuals, they have to fulfil a second none the less important function, namely to compensate side-effects such as too long lines or too long columns.

The individuals to be mutated were chosen randomly by an equally distributed function. In the following each of the mutation approaches will be described.

Algorithm 9: Horizontal flop

```
input  : solution: the individual to be mutated
output: newSolution: the mutated individual
begin
    Create newSolution;
    maxLine ← number of lines;
    Compute splitHorizontal;                                // randomly chosen
    for i ← splitHorizontal; i < maxLine; i++ do
        row = solution.getLine(i);         // copy the bottom half to the top
        newSolution.add(row);
    end
    for i ← 0; i < splitHorizontal; i++ do
        row = solution.getLine(i);         // copy the top half to the bottom
        newSolution.add(row);
    end
    return newSolution;
end
```

4.3.1 Horizontal flop

The *horizontal flop* (HFM), visualised in Alg. 9, is the mutating equivalent of HBX. In the case of the mutation the parent individual is horizontally split into two parts between two rows of shreds dividing the `solution` into an upper half and a lower half, see the example in Fig. 4.3a.

As for the recombination the splitting position is chosen randomly on the base of a pseudo Gaussian distribution function.

The lower part of the parent is then copied to the upper part of the descendant and the same is done for the other half. Thus all left-right and top-bottom relations are inherited to the `newSolution` except the relations along the splitting line. Roughly speaking the parent individual made a flip-flop.

The complexity of this mutation is dependent from the number of lines of `solution`, which can be up to n lines, using n shreds. Thus in the worst case the complexity of this mutation is $O(n)$.

Algorithm 10: Vertical flop

```
input  : $solution$: the individual to be mutated
output: $newSolution$: the mutated individual
1  begin
2      Create $newSolution$;
3      $maxLine \leftarrow number\ of\ lines$; $maxColumn \leftarrow number\ of\ columns$;
4      Compute $splitHorizontal$;                              // randomly chosen
5      for $i \leftarrow 0$; $i < maxLine$; $i{+}{+}$ do
6          $pos \leftarrow 0$;
7          for $j \leftarrow splitVertical$; $j < end\ of\ line$; $j{+}{+}$ do
8              $row$.add($solution$.getLine($i$).getColumn($j$));   // copy left to right
9              $pos{+}{+}$;
10         end
11         Fill space $maxColumn - end\ of\ line$ with $virtualShred$;
12         for $j \leftarrow 0$; $j < splitVertical$; $j{+}{+}$ do
13             $row$.add($solution$.getLine($i$).getColumn($j$));   // copy right to left
14         end
15         $newSolution$.add($row$);
16     end
17     return $newSolution$;
18 end
```

4.3.2 Vertical flop

The *vertical flop* (VFM), see Alg. 10, is the logical add-on to the other three operators already described. Now, as for the VBX, the splitting line moves vertically along a column of shreds, dividing the `solution` into a left and right half. This line is again chosen randomly taking the parents longest column into account.

For each line the `newSolution` inherits the left side from the parents right half, as shown in line 7 to 10, and then vice versa, see line 12 to 14.

Due to the fact that the lines of `solutions` in general do not share the same length, the left side of the descendant is filled with virtual shreds, as done in line 11. Otherwise the right half would be shifted to the end of each line and thus the top-bottom relations could not be passed on from the parent to the descendant, as visualised in Fig. 4.3b.

All the n shreds must be copied in their new order to `newSolution`, which leads to a complexity of $O(n)$.

Algorithm 11: Switch two

input : *solution*: the individual to be mutated; *repeat*: the number of switches
output: *newSolution*: the mutated individual

```
1  begin
2      for $i \leftarrow 0$; $i < repeat$; $i{+}{+}$ do
3          Choose $shredA$ randomly;
4          Choose $shredB$ randomly;
5          Get $positionA$ of $shredA$;
6          Get $positionB$ of $shredB$;
7          $positionA \leftarrow shredB$;
8          $positionB \leftarrow shredA$;
9      end
10     return $newSolution$;
11 end
```

4.3.3 Switch two

Mainly the *switch two* (S2M) mutation operator is the most simple but also most flexible one. The basic idea is to randomly swap two shreds. This operation can, however, be repeated up to 10 times.

Due to efficient datastructures an application of this operator can be performed in $O(1)$.

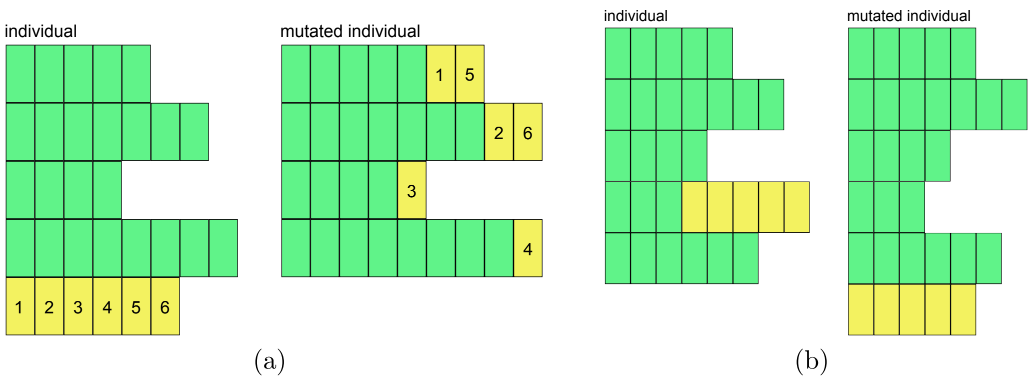


Figure 4.4: (a) An example for the break column; (b) an example for the break line

Algorithm 12: Break column

```
   input  : $solution$: the individual to be mutated
   output: $newSolution$: the mutated individual
 1 begin
 2     Create $newSolution$;
 3     $maxLine \leftarrow number\ of\ lines$;
 4     Compute $lines$;          // number of lines to remove at most one-fifth
 5     for $i \leftarrow 0;\ i < (maxLine - lines);\ i{+}{+}$ do
 6         $row = solution$.getLine($i$);        // copy all lines except last $lines$
 7         $newSolution$.add($row$);
 8     end
 9     $pos \leftarrow 0$;                        // add remaining shreds to the copied lines
10     for $j \leftarrow i;\ j < maxLine;\ j{+}{+}$ do
11         for $k \leftarrow k;\ k < end\ of\ line;\ k{+}{+}$ do
12             $shred \leftarrow solution$.getLine($j$).getColumn($k$);     // get current shred
13             $newSolution$.getLine($pos$).add($shred$);    // add at end of line $pos$
14             $pos{+}{+}$;
15             if $pos == (maxLine - lines)$ then
16                 $pos \leftarrow 0$;                                        // reset $pos$
17         end
18     end
19     return $newSolution$;
20 end
```

4.3.4 Break column

The *break column* (BCM) helps to control a side-effect of a recombination in that case of the 2DERX, which creats individuals with short but many lines.

Within this mutation the bottom fifth of all rows (or less) are selected and in the following removed, see line 4 of Alg. 12. The shreds of these lines are added to the end of the remaining lines. All the shreds from the bottom lines are taken, see code-lines 10 and 11, and added in turn to the top lines while every new shred appends a line beneath the previous, code-line 13. If the last line of `newSolution` is reached the process restarts with the first line, see code-lines 15 and 16. An example is shown in Fig. 4.4a

Up to one-fifth of n shreds are copied within this mutation, which results in a complexity of $O(n)$.

Algorithm 13: Break line

```
input  : $solution$: the individual to be mutated
output: $newSolution$: the mutated individual
1  begin
2      Create $newSolution$;
3      $longestLine \leftarrow position\ of\ longest\ line$;
4      $maxColumn \leftarrow longestLine$.size();
5      Compute $split$;                                        // randomly chosen
6      for $i \leftarrow 0$; $i < longestLine$; $i{+}{+}$ do
7          $row = solution$.getLine(i);                        // copy until longest line
8          $newSolution$.add($row$);
9      end
10     $row$ = Shreds of $longestLine$ from begin until $split$;
11     $newSolution$.add($row$);                   // keep first part of shreds in line
12     for $i \leftarrow longestLine + 1$; $i < number\ of\ lines$; $i{+}{+}$ do
13         $row = solution$.getLine($i$);              // copy from longest line to end
14         $newSolution$.add($row$);
15     end
16     $row$ = Shreds of $longestLine$ from $split$ until end;
17     $newSolution$.add($row$);              // add second part of shreds to the end
18     return $newSolution$;
19 end
```

4.3.5 Break line

The *break line* (BLM) was designed because first tests showed that especially HBX and VBX created `solutions`, whose lines grew longer over the time. This is certainly because of the best fit heuristic that often adds remaining shreds to the end of lines. But with the help of the BLM this problem is under control.

The idea is to find the longest line of the parent individual and choose a point randomly where this line should be split apart. Again the pseudo Gaussian approach from HBX was used for line 5 of Alg. 13.

Then the first half of `solution` is copied to `newSolution` until the longest line is reached, see lines 6 to 9. From this line only the shreds from the beginning until the splitting point are added to `newSolution` as indicated in line 10. Afterwards the rest of the lines are copied. In the end a new line is created where the second half of the longest line from the breaking point until the end is added, see line 16. This makes `newSolution` one line longer but with less columns, see Fig. 4.4b.

In the worst case there are $n-1$ lines using n shreds, one line must have two shreds, ohterwise no line could be broken. This results in a complexity of $O(n)$.

4.4 Improvement

As a common proceeding a GA is extended by a heuristic, which is used to improve the quality of the individuals. As mentioned above this extension of a GA is called a *memetic algorithm*. The idea is to use fast and simple algorithms every few generations to fathom new solution opportunities. Thus restrictions that could emerge from the population based search can be mended with the point based search of the heuristics, see 3.2.

Important is to turn the attention on the phrase fast and simple, because the improvement strategies are usually applied to many individuals and thus too complex approaches would have devastating effects concerning the computation time.

The already mentioned VNS with VND, introduced in [23] and advanced in [28], was used as an improvement strategy for the GA. The moves and neighbourhoods created for this VNS, taken from [28], will be described in the following, while the general working principle of VNS is outlined in Sec. 3.3.2. At first there must be a differentiation made between these moves:

SwapMove swaps two different shreds i and j, with $i, j \in S$.

ShiftMove allows to shift a rectangular region of shreds within the solution. Therefor the position and size of the region, the shifting direction and distance must be defined.

These moves lead to the following seven neighbourhood structures:

N_1: One swap move is applied within the current solution.

N_2: Within this neighbourhood structure one shred is shifted either horizontally or vertically.

N_3: This structure shifts a single row or column of defined length.

N_4: In that case the length and width of the rectangle of shreds to be shifted can be chosen arbitrarily.

N_5: One single shred is first shifted horizontally and then vertically.

N_6: This structure allows to shift a square of shreds first horizontally and then vertically.

N_7: Contrary to N_6 now a rectangle of shreds is shifted first horizontally and then vertically.

To fulfil the fast and simple criteria mentioned above only three of the large pool of possible neighbourhood operations are utilised. These specific moves according to their implementation are described in Sec. 5.3.

The VNS is then applied over its complete runtime to ten percent of the best individuals of the current population.

To further reduce the the negative effect of VNS on computation time, this local improvement phase is only applied to the individuals of every five thousandth population. For comparisons of GA configurations with and without the application of VNS to individuals see Sec. 6.

4.5 Final improvement

Having an improvement heuristic already implemented it would of course be suggestive to use it again at the end of the GA. The *ImproveSolutions($P(t)$)* is called many times every few thousand generations within the reproduction cycle. In prospect of an even better improvement, it may be convenient to allow a longer search time for the *ImproveFinalSolution($P(generations)$)* execution of VNS but to apply it only to the best individual of the last population. The concrete moves based on the previously described neighbourhood structures are mentioned in Sec. 5.3.

Although the local improvement phase *ImproveSolutions($P(t)$)* was only applied during one test configuration, this final improvement is performed each run.

CHAPTER 5

Implementation

The whole GA, described in Sec. 4, was then implemented in JAVA 1.6 and extends an already existing framework. This framework consists of a graphical user interface (GUI) and a command line interface (CLI). Moreover does the framework handle the data representation of the shredded pages and computes the EEF values, see Sec. 2.2.

The GUI is able to visualise a strip or cross-cut shredded page and the implemented algorithms can be consecutievly applied to one instance. Because of the visualisation it is possible to visually check the quality of a reconstructed document.

The CLI on the other hand is designed for the purpose of testing. It allows to apply one algorithm to one instance at a time and the progress of the algorithm can be controlled on the command line by logging messages. All the tests for this thesis, see Sec. 6, were done with the help of the CLI.

In the implemented framework a possible solution is saved as a two dimensional Java ArrayList, in which all lines are linked to one ArrayList and the columns of each line are stored in an ArrayList of their own. All shreds have a unique number, which are saved in the column ArrayLists as integers. Exactly this approach was taken for the GA to represent the genotype of each individual. The shred images linked to the integer values on the stored positions would conform to the phenotype.

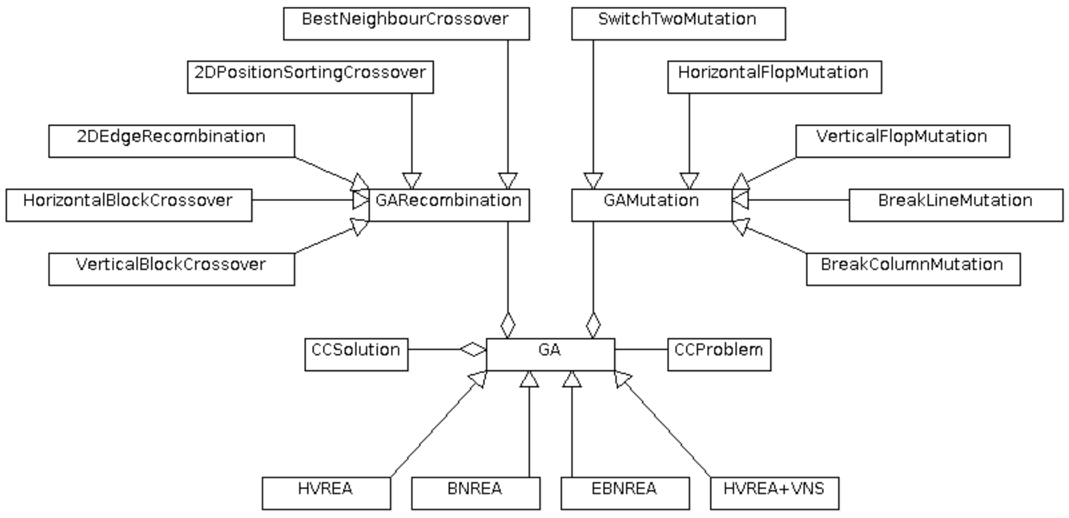


Figure 5.1: A schematic illustration of the GA components and their dependencies

5.1 Module Structure

In the following a schematic illustration of the GA components and the position of the GA within the framework will be given.

Fig. 5.1 shows the GA components and the connections between them. The center builds the abstract `GA` class itself, which is a JAVA implementation of Alg. 3. The test configurations `HVREA`, `BNREA`, `EBNREA` and `HVREA+VNS`, from Sec. 6, are derived from that class.

The input, i.e., the shredded document, is represented by the class `CCProblem`. Within this class the shreds and the EEF values for all shreds compared with each other are stored. The output, i.e., the reconstructed document or in case of the GA the individual, is handled by the `CCSolution` class. For each individual an instance of this class is created, which stores for instance the fitness of the individual and the position of each shred within the solution.

The crossover operators are all derived from the `GARecombination` interface, while the `GAMutation` interface builds the base for all mutation operators. The classes derived from these two interfaces are implementations of the operators described in Sec. 4.

In Fig. 5.2 the position of the GA within the whole framework can be seen. The `ACO`, `GA`, `VNS` and `ILP` components represent the implementations of the other recon-

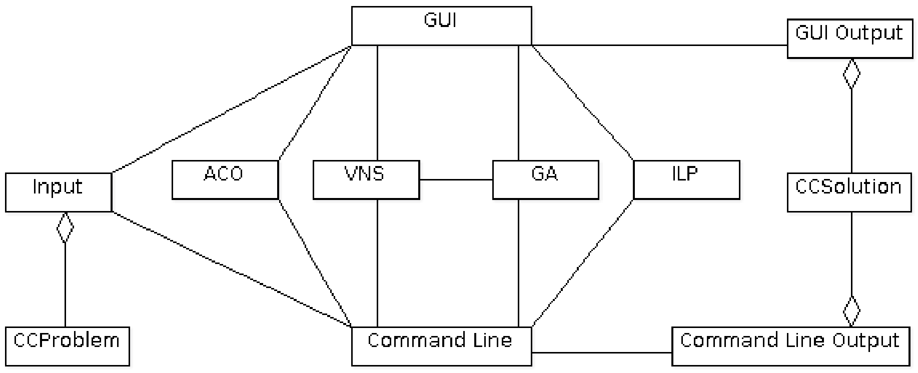


Figure 5.2: A framework illustration and the connection between its components

struction approaches. The connection between the `GA` and `VNS` components hints at the collaboration between these two algorithms.

These components can be used in the surrounding of the `GUI` or the `CommandLine`. All the algorithms use the `CCProblem` class as input. The output is, as for the GA, handled by the `CCSolution` class. In case of the `GUI` a graphical visualisation of the reconstructed document is possible based on the position informations stored in `CCSolution`. The `CommandLine` on the other hand uses the fitness value, i.e., the overall error made according to EEF, from `CCSolution` as output.

5.2 Starting the Framework

To compile and run both the GUI and the CLI the JAVA based program ANT is used. The following several parameters control, which instance is taken or define the cutting pattern and are always called from the command line:

guiCrossCut starts the GUI.

crossCut*Algorithm* starts *Algorithm* using the CLI setup.

-DinputFile selects the document, which is cut apart and then reconstructed. The file format must be *.png.

-DnumOfShredsX sets the horizontal number of shreds.

-DnumOfShredsY sets the vertical number of shreds.

-DEAGenerations sets the number of generations of the GA.

-DEAIndividuals sets the population size of the GA.

For example the line:

```
ant crossCutEA3 -DinputFile p03 -DnumOfShredsX 8 -DnumOfShredsY 7
-DEAGenerations 8000 -DEAIndiduals 200
```

starts the GA type BNREA, see Sec. 6, with the instance p03, using a 8x7 cutting pattern and for the GA the number of generations is set to 8.000 with a population size of 200 individuals. Using `guiCrossCut` for instance instead of `crossCutEA3` would start the GUI.

5.3 VNS Settings

The neighbourhood structures described in Sec. 4.4 were then implemented in the framework and used for the VNS call within the GA.

Within the *ImproveSolutions(*$P(t)$*)* method the VNS is then applied to the ten percent of the best individuals of the current population. For the VNS used in that case the following fast and simple moves are used:

SimpleSwap swaps two different shreds within the solution.

SimpleShift shifts a single shred either horizontally or vertically.

SimpleBlockShift takes a square block and shifts it either horizontally or vertically.

For the *ImproveFinalSolution(*$P(generations)$*)* to further enlarge the improvement potential additional more complex moves are applied:

SimpleDoubleShift can shift a single shred both horizontally and vertically.

SimpleDoubleBlockShift shifts a square block both horizontally and vertically.

RectangleBlockShift takes a rectangle block and shifts is either horizontally or vertically.

RectangleDoubleBlockShift shifts a rectangle shaped block both horizontally and vertically.

CHAPTER 6

TESTS

To test the implemented operators ten sampling instances (p01–p10) were chosen, see App. B. Since the main focus of this thesis lies on the reconstruction of shredded text documents all of these ten instances consist of type writer written text. Nevertheless, some of them also contain small images and figures. Moreover not all of them have the shape of the common DIN A4 format, namely p06, p07 and p08.

For the test the documents were horizontally and vertically cut apart using 8, 11 or 14 cuts, which leads to nine different cutting patterns for each instance from 9x9 snippets up to 15x15 snippets.

6.1 CONFIGURATIONS

Based on preliminary tests configurations were figured out such that the crossover and mutation operators interact in a promising manner. Unluckily it turned out that the *2D position sorting crossover* failed to improve the initial population at all and thus it was ignored during the final test phase. For the remaining operators the following configurations were settled.

All the configurations have in common that a population size of 300 individuals is used over a period of 30.000 generations. At the end of the GA itself, i.e., after the 30.000 generations, the VNS from [28] is applied to the best individual of the last population using the moves described in Sec. 4.5.

HVREA

This setup tests the *horizontal block crossover* combined with the *vertical block crossover*. While both operators create two descendants out of two parents only the best offspring from each crossover was chosen to improve the quality of the next population. On 25% of all individuals mutation was applied whereas the concrete probability for choosing the according mutation operator is shown in Tab. 6.1.

Table 6.1: The mutation probabilities for the HVREA

HFM	**VFM**	**BLM**	**S2M**
5%	5%	10%	5%

BNREA

With this configuration the *best neighbour crossover* is tested. Because it is the only crossover used in this setup both descendants of the operator were used for the next population. A mutation rate of 25% was applied, while the concrete probabilities can be found in Tab. 6.2.

Table 6.2: The mutation probabilities for the BNREA

HFM	**VFM**	**S2M**
5%	15%	5%

EBNREA

The purpose of this configuration is to test the effect of the *2D edge recombination*. Because of the side-effects of 2DERX, the columns get normally too long, a high mutation rate for BCM is needed and the BNX was also used to make the individuals more compact. Moreover in the first 8.000 generations only the BNX with the best of both descendants creates the new population. This is again done to smooth the side-effects of 2DERX. The idea is to create a better starting setup for 2DERX, which proved very efficient due to preliminary tests. After this beginning phase the 2DERX, which can only create one child, and the BNX with the better offspring together build the new population. An overall mutation rate of 35% is chosen with the specific probabilities shown in Tab. 6.3.

Table 6.3: The mutation probabilities for the EBNREA

BLM	BCM	S2M
10%	20%	5%

HVREA+VNS
This configuration intends to show the possibilities of document reconstruction using the GA created for this thesis with additional help from the VNS. The setup parameters from the original HVREA are taken but the population size is expanded to 700 individuals and the reproduction cycle lasts 70.000 generations. Moreover VNS, using simpler but faster moves like described in Sec. 4.4, is applied every 5.000 generations to the best 10% of individuals of the current population. In the end VNS is used to improve the best individual of the last population in the usual way.

Due to the fact that this approach is more time consuming and thus the probability of finding better solutions is much higher, this configuration can not be seen as a competitor to the three configurations mentioned before. It is, however, interesting to see how regularly performed local search phases influence the performance of this GA setting.

6.2 Results

Each configuration was tested with 30 runs on all 90 instances, i.e., the ten sample pages with nine cutting patterns each. Then the mean percentage gap for each instance and each configuration was computed. Moreover a student t-test was applied comparing all the configurations pairwise.

Detailed test results can be found in App. A. Tab. A.1 shows the results for each instance after the end of the GA and before the VNS call, while the results after the VNS call can be found in Tab. A.2. For comparison the computational results of instances p01–p05 using the *ant colony optimisation* (ACO) from [29] are also shown in these tables.

At first glance it can be seen that some instances can be solved true to original by all the tested configurations. Excluding these instances a hierarchy arises with the HVREA+VNS as the best and the BNREA as the worst configuration, while the HVREA usually remains better than the EBNREA. Bringing the ACO into this hierarchy it performs a bit better than the HVREA without the VNS at the end but

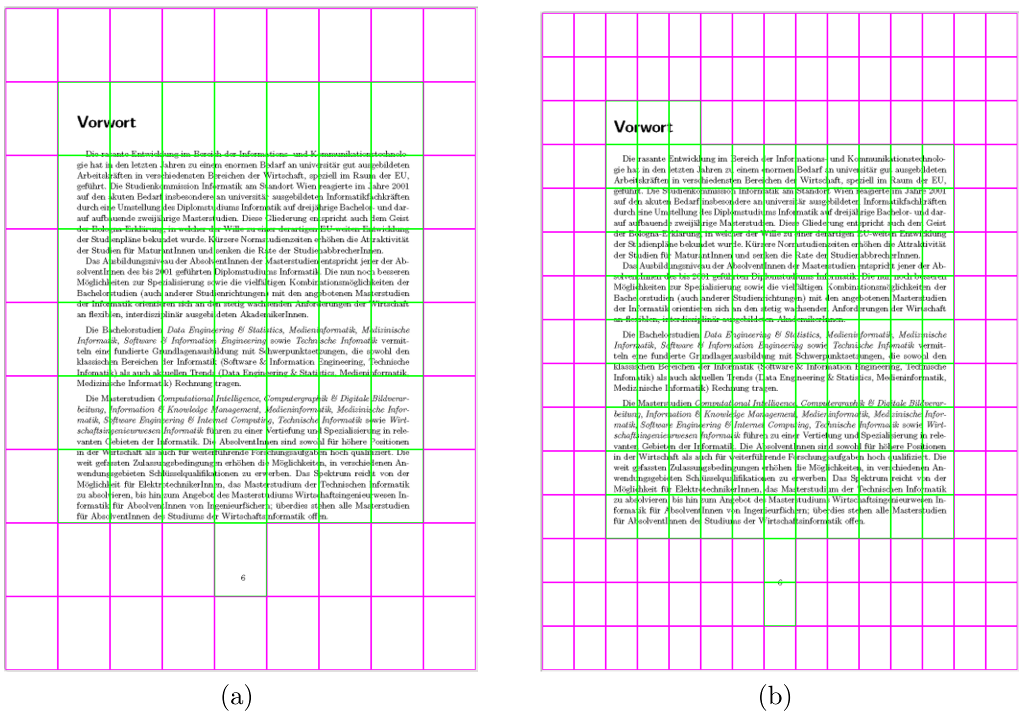


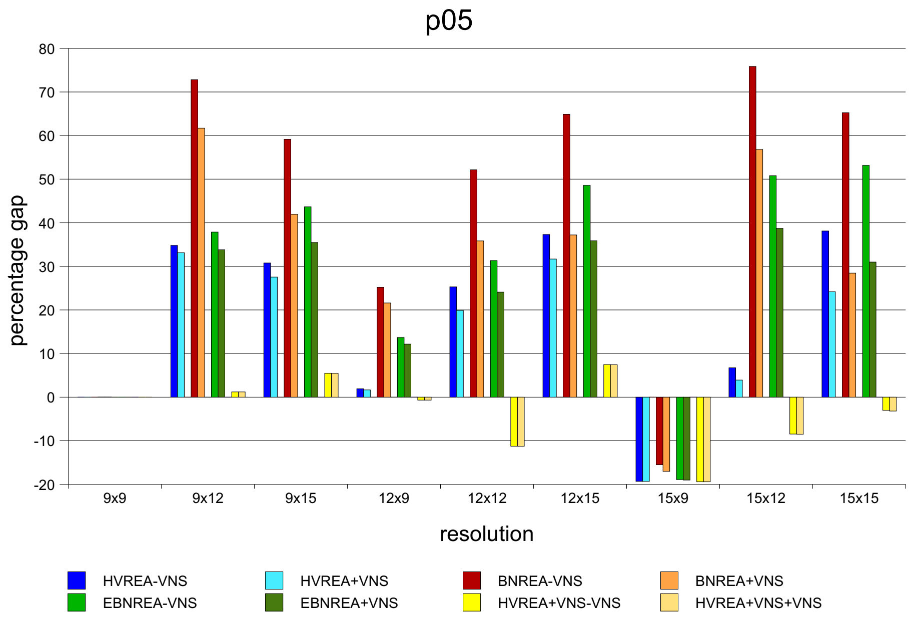
Figure 6.1: (a) Cutting p01 into 9x9 shreds is easy to solve
(b) while cutting it into 15x15 shreds it becomes one of the most difficult instances

a bit worse than the HVREA with the VNS in the end. The ACO is usually better than the EBNREA without the VNS and equal to it with the VNS at the end.

While the *error estimation function* (EEF) from Sec. 2.2 performs well for text documents, it does not take into account that the error induced by the original document is the minimal error. Because of the massive usage of local improvement and the long period of time available HVREA+VNS was able to find an adjustment of shreds with an EEF value smaller than the original one for many instances, which is represented by a negative percentage gap in the tables.

Comparing the results it appears that for some documents a reconstruction is easier than for others. While samples like p02, p03 and p04 can be reconstructed efficiently others like p01, p08 and p09 are difficult to solve.

On the other hand the reconstruction also depends on the cutting pattern. A very good example for this is p01 which can be solved easily if the cutting pattern is 9x9 shown in Fig. 6.1a, while the pattern 15x15, Fig. 6.1b, is one of the most difficult to solve. Moreover for all the instances regarding this sample no adjustment was found with an EEF value smaller than the value of the original document.



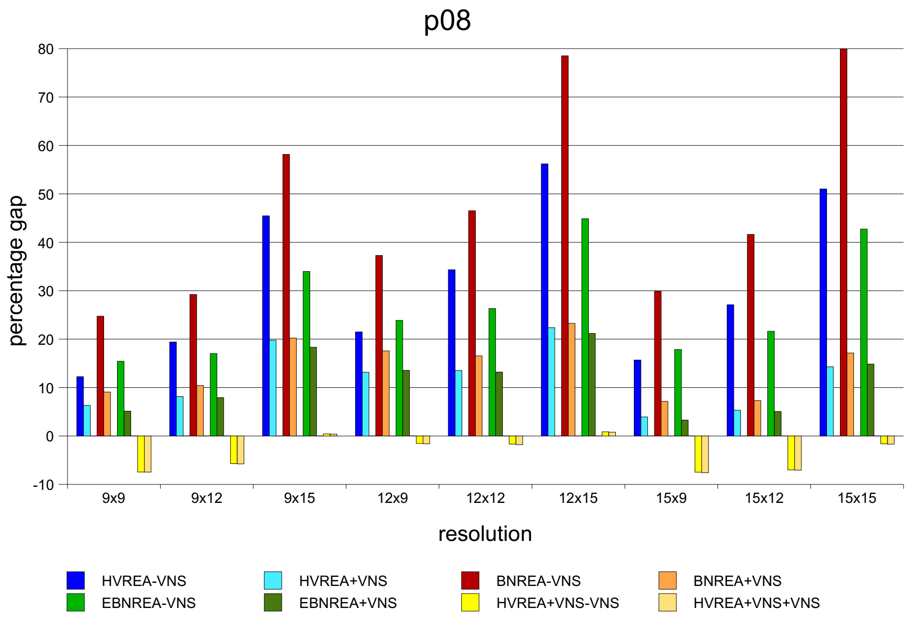
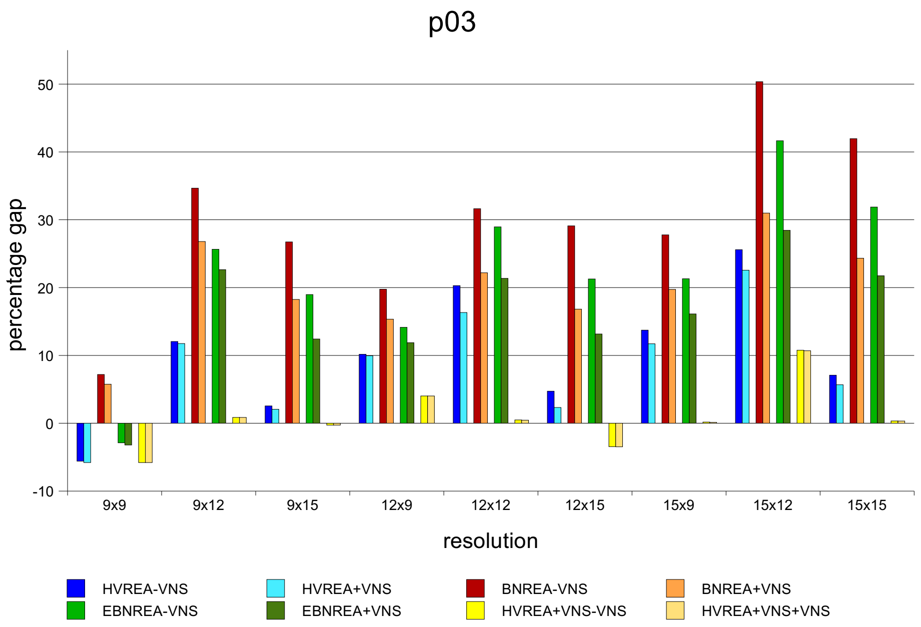


Figure 6.2: Reconstructing these two samples depends on the cutting pattern



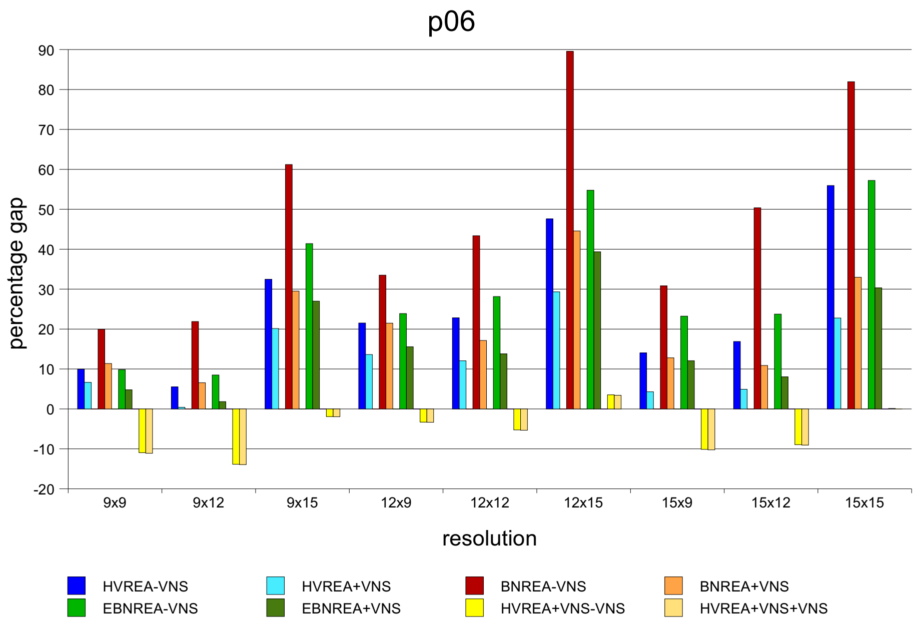


Figure 6.3: These instances are all equally difficult to solve

Comparing the quality of the solutions of all configurations and all cutting patterns for one sample confirms this discussion. The two charts in Fig. 6.2 show two samples, where different patterns lead to either easily or difficult reconstructable instances of the same document. In these diagrams two neighbouring bars stand for the same configuration but the left bar always represents the performance of the GA alone and the right with the VNS applied in the end.

For sample p05 and a pattern of 9x9 cuts all configurations were able to perfectly reconstruct the original document. Using the 15x9 pattern an adjustment with a smaller EEF value is found by all configurations. In all the other cases this instance proves to be difficult to solve.

For sample p08 the HVREA+VNS can find better adjustments in seven out of nine situations. But all the other configurations perform bad if this document is cut apart 15 times vertically, while the VNS helps a lot to solve the other cutting patterns.

In Fig. 6.3 two samples are shown, whose reconstructions do not depend that much on the cutting pattern. For sample p03 the VNS can barely find any improvement, while the VNS helps a lot for p06.

Another point of interest is the behaviour of the crossover operators during the optimisation process. In other words how fast do these operators converge? To discuss this question some instances were taken and analysed in reference to the convergence of the crossover operators.

In that case the behaviour of HVREA+VNS solving four different instances was observed alone, because the higher amount of generations within this configuration made it difficult to compare it to the other approaches. The diagram in Fig. 6.4 indicates that the great advancement of improvement is done during the first 25.000 generations. There is to note that all 5.000 generations VNS was applied and after 10.000 generations the improvement steps can clearly be seen in the chart. This means that the GA alone is not able to create better solutions but the further improvement is possible because the newly created solutions build a better base for the VNS.

In the two instances shown in Fig. 6.5, the operators behave alike. The best configuration is the HVREA, closely followed by the EBNREA and the BNREA placed last. The right side of the x-axis marks the end of the GA and the VNS applied afterwards.

A typical behaviour, which can also be seen here, is that the VNS improves the BNREA the most. While this configuration without the VNS is always defeated by the others, the VNS helps to significantly improve the BNREA and in some cases there is no significant difference to the EBNREA after the VNS improvement according to the student t-test, see Tab. A.2.

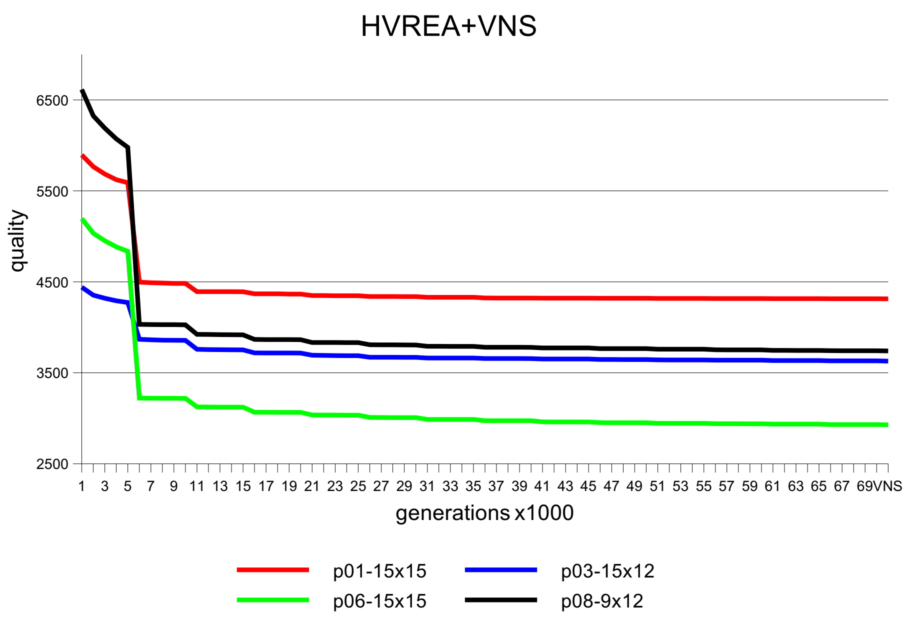
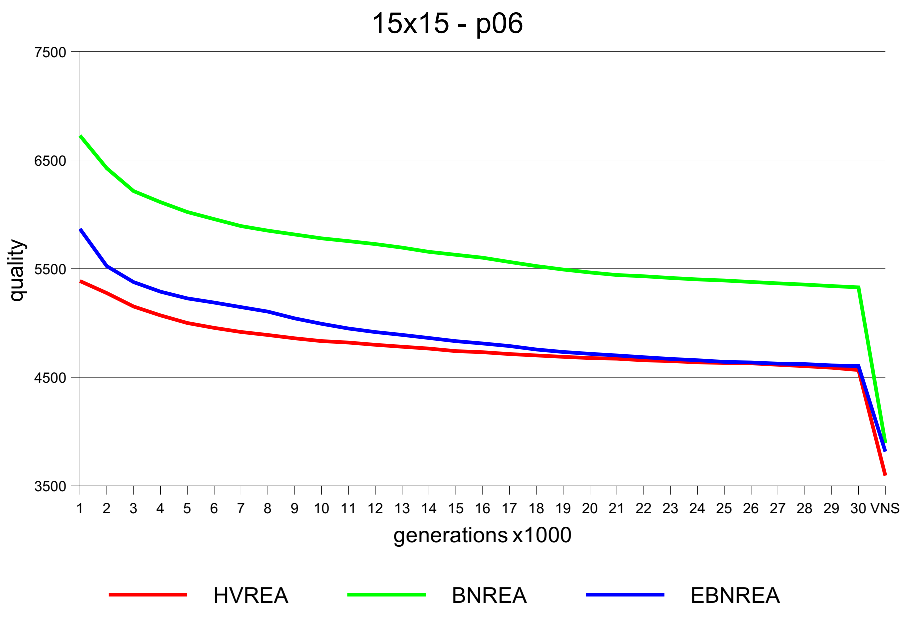


Figure 6.4: The convergence of the HVREA+VNS configuration

Comparing the convergence of the operators the HVREA clearly converges the fastest, followed by the BNREA while the EBNREA shows the slowest convergence. This leads to the consideration that for the EBNREA more than 30.000 generations would emerge in an even better result using 300 individuals, while the other two operators do not need a longer reproduction cycle.

The diagram in Fig. 6.6a shows a situation, in which the EBNREA proves to be better than the HVREA, both without the VNS. It is also one of the most difficult instances to solve for the BNREA, again without the VNS. Taking the VNS into account the result for all the operators is nearly the same. Again the BNREA benefits the most from the applied VNS.

The chart in Fig. 6.6b shows the behaviour of the crossover operators applied to the 9x9–p09 instance. This instance is the opposite of the diagram previously discussed. Here the HVREA performs much better than the other two. Moreover it is an instance, which can barely be improved by the VNS, especially for HVREA.



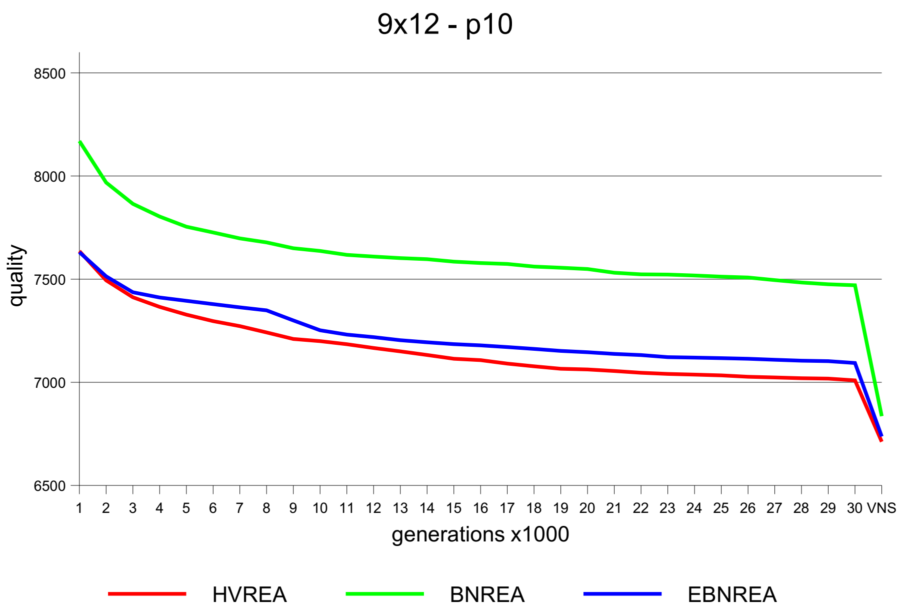
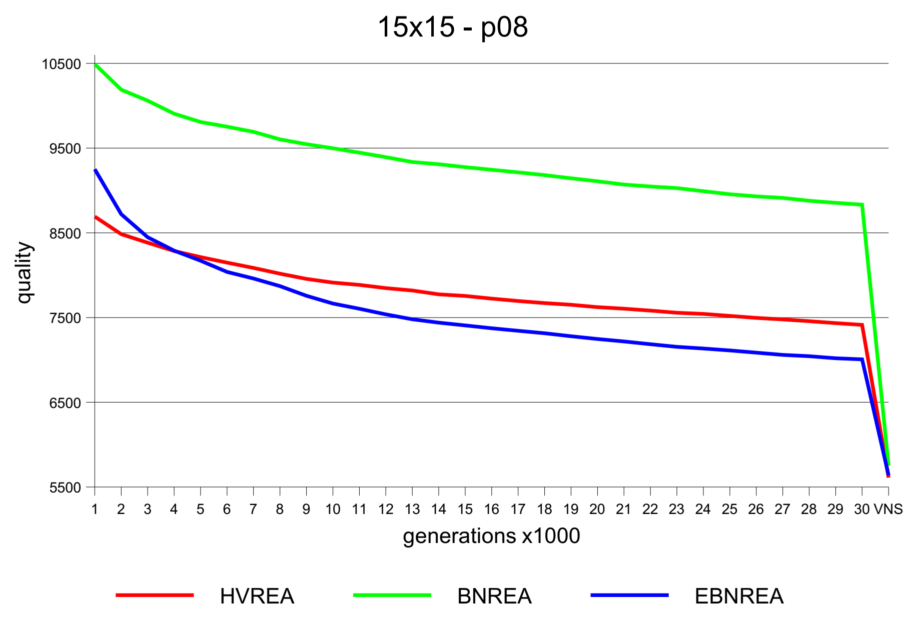
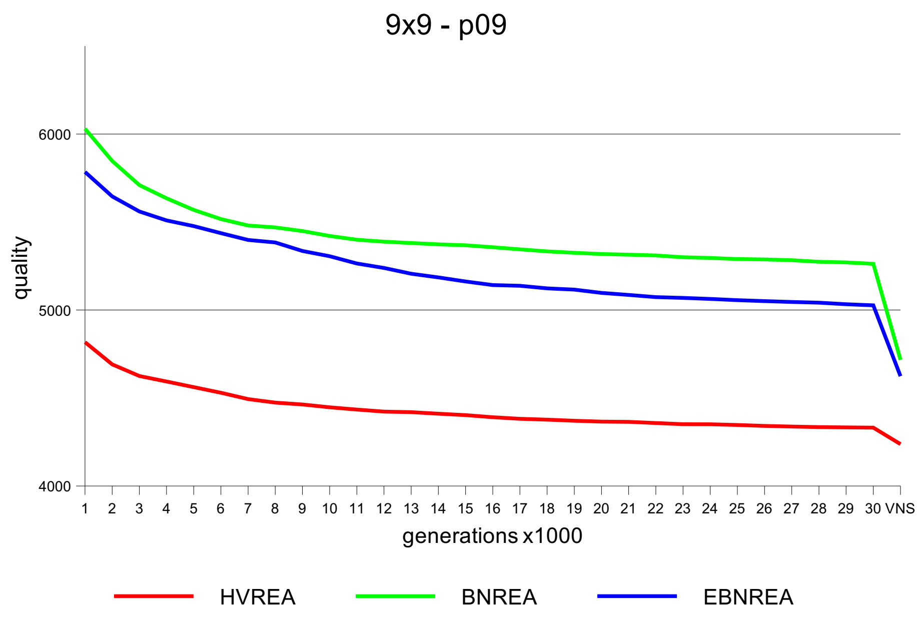


Figure 6.5: The operators behave alike solving these two instances



(a)



(b)

Figure 6.6: Two instances with completely different behaviour

CHAPTER 7

Conclusion and Future Work

In this master thesis the problem of reconstructing cross-cut shredded documents was presented and a *genetic algorithm* (GA) designed to solve this problem. Due to the definition this problem is two dimensional and asymmetric. One ambition was to extend approved crossover and mutation operators in a way that they can be applied to this problem.

While this was possible for the *one-point crossover* and the *edge recombination*, the two dimensional approach of the *position sorting crossover* failed to meet the requirements.

To evaluate a reconstructed document a so-called *error estimation function* (EEF) was introduced, which computes the error made, when placing two shreds next to each other. While this function performs well, one problem is that the error induced by the original document is not necessarily the minimal error, i.e., there might be an adjustment of shreds with an EEF value smaller than the value of the original document.

The implementation of the designed GA was then tested on ten samplings with nine different cutting patterns each and always using an evaluation according to EEF. One observation of the test results was that even akin looking documents proved to be both easy and difficult to reconstruct. Another point is that the result of the reconstruction often depends on the cutting pattern, i.e., the number of cuts horizontally and vertically.

The results have shown that with a small computational effort a reconstruction of at least two thirds of a document is possible. Having more computational power and taking the usage of local improvement into account a nearly perfect reconstruction is always possible. Furthermore, it was possible to outperform a previously introduced ant colony optimisation approach using one of the GA variants presented within this work.

Nevertheless all the samples introduced represented one single printed page. The GA was never tested on multi page samples but it would be interesting see the results of these tests.

The introduced EEF only works with grayscale values and is designed for the field of reconstructing text documents. Extending the EEF to the RGB colour space would surely help to reconstruct images.

Another approach would be to formulate a new and different EEF, which is based on the matters of *optical character recognition* to solve text documents or an EEF for images based on the methods of *image processing* and *content-based image retrieval.*

A possible extension for the framework could be to allow human interaction in a way that the optimisation process can be paused and the best solution for instance is shown. As soon as two shreds in such a solution are correctly positioned with respect to each other, the user can stick them together such that the optimisation can be continued handling these two shreds as one large shred.

APPENDIX A

Results

Table A.1: The mean percentage gaps over 30 runs and corresponding standard deviations for the tested configurations **without** the application of VNS in the end are listed here. For comparisons the available ACO results from [29] are also presented. Moreover the values in columns p correspond to the student t-test with 5% error level and indicate the relation between two neighbouring configurations.

				HVREA			BNREA			EBNREA			HVR+VNS		ACO	
	x	y	orig	mean	dev	p	mean	dev	p	mean	dev	p	mean	dev	mean	dev
instance p01	9	9	2094	0.0%	(0.0)	$\approx$	0.0%	(0.0)	$\approx$	0.0%	(0.0)	$\approx$	0.0%	(0.0)	0.0%	(0.0)
	9	12	3142	41.6%	(7.0)	$>$	34.6%	(7.7)	$>$	24.2%	(6.3)	$>$	6.9%	(5.7)	21.0%	(4.8)
	9	15	3223	46.9%	(6.1)	$>$	37.6%	(6.9)	$>$	25.3%	(12.0)	$>$	6.3%	(6.3)	30.7%	(2.1)
	12	9	2907	42.2%	(8.2)	$>$	27.7%	(8.9)	$>$	18.7%	(10.9)	$>$	0.5%	(7.6)	25.2%	(3.2)
	12	12	3695	41.2%	(6.5)	$>$	36.9%	(6.9)	$>$	27.6%	(4.8)	$>$	6.6%	(2.3)	27.5%	(2.9)
	12	15	3825	56.3%	(7.5)	$>$	50.1%	(5.1)	$>$	34.4%	(5.4)	$>$	10.2%	(2.0)	32.7%	(3.2)
	15	9	2931	39.0%	(5.9)	$>$	32.5%	(5.4)	$>$	1.6%	(6.2)	$\approx$	0.0%	(0.0)	29.1%	(4.1)
	15	12	3732	50.5%	(7.1)	$>$	40.8%	(6.3)	$>$	27.9%	(8.8)	$>$	9.0%	(4.3)	32.5%	(2.6)
	15	15	3870	53.8%	(7.2)	$>$	43.8%	(6.5)	$>$	39.9%	(5.0)	$>$	11.5%	(2.2)	33.2%	(3.2)
instance p02	9	9	1434	4.0%	(17.7)	$>$	-10.4%	(7.5)	$>$	-23.4%	(6.2)	$>$	-29.7%	(0.6)	4.8%	(4.6)
	9	12	1060	57.3%	(20.6)	$>$	38.1%	(13.8)	$>$	7.9%	(8.1)	$>$	0.0%	(0.4)	35.3%	(5.1)
	9	15	1978	27.7%	(10.7)	$>$	11.6%	(8.4)	$>$	0.4%	(3.5)	$>$	-11.0%	(0.9)	21.7%	(3.6)
	12	9	1396	-3.8%	(10.4)	$>$	-12.9%	(5.4)	$>$	-17.7%	(9.2)	$>$	-30.4%	(1.4)	13.4%	(4.2)
	12	12	1083	34.6%	(15.2)	$>$	10.9%	(6.2)	$>$	5.9%	(10.2)	$>$	-0.2%	(1.1)	27.7%	(5.0)
	12	15	1904	14.5%	(9.0)	$>$	-3.6%	(3.5)	$<$	3.0%	(6.9)	$>$	-10.7%	(2.0)	19.7%	(3.5)
	15	9	1658	15.4%	(7.8)	$>$	9.4%	(7.1)	$>$	-6.5%	(8.4)	$>$	-16.3%	(2.0)	13.8%	(4.0)
	15	12	1503	25.5%	(10.9)	$>$	15.1%	(8.7)	$>$	9.2%	(10.9)	$>$	0.1%	(0.5)	19.8%	(3.3)
	15	15	2283	10.4%	(6.1)	$>$	6.3%	(4.2)	$\approx$	3.9%	(6.9)	$>$	-6.3%	(2.3)	12.6%	(1.7)

				HVREA			BNREA			EBNREA			HVR+VNS		ACO	
	x	y	orig	mean	dev	p	mean	dev	p	mean	dev	p	mean	dev	mean	dev
instance p03	9	9	2486	7.2%	(7.0)	>	-2.9%	(4.1)	>	-5.6%	(4.7)	≈	-5.8%	(5.0)	6.7%	(4.8)
	9	12	2651	34.7%	(10.1)	>	25.6%	(7.3)	>	12.1%	(8.5)	>	0.9%	(3.7)	30.0%	(3.6)
	9	15	2551	26.7%	(7.0)	>	19.0%	(5.4)	>	2.6%	(4.3)	>	-0.3%	(1.3)	18.0%	(2.6)
	12	9	3075	19.8%	(6.1)	>	14.1%	(5.6)	>	10.2%	(5.0)	>	4.0%	(1.2)	11.6%	(2.9)
	12	12	3377	31.6%	(8.4)	≈	29.0%	(6.1)	>	20.3%	(5.9)	>	0.5%	(4.7)	23.3%	(2.5)
	12	15	3313	29.1%	(5.9)	>	21.3%	(6.3)	>	4.7%	(7.4)	>	-3.5%	(0.4)	16.3%	(2.1)
	15	9	3213	27.8%	(6.5)	>	21.3%	(4.6)	>	13.7%	(6.3)	>	0.2%	(2.5)	13.6%	(2.3)
	15	12	3278	50.4%	(5.2)	>	41.6%	(7.0)	>	25.6%	(11.6)	>	10.8%	(4.3)	36.3%	(2.1)
	15	15	3308	42.0%	(5.3)	>	31.9%	(5.2)	>	7.1%	(7.3)	>	0.3%	(1.9)	27.0%	(1.4)
instance p04	9	9	1104	21.4%	(13.2)	>	5.7%	(13.9)	>	-13.6%	(14.6)	>	-25.0%	(10.8)	18.4%	(7.2)
	9	12	1463	13.7%	(11.0)	>	6.2%	(8.9)	>	-3.0%	(9.8)	>	-15.1%	(3.0)	15.6%	(4.5)
	9	15	1589	-0.8%	(5.4)	>	-7.3%	(4.9)	>	-14.2%	(8.7)	>	-24.0%	(1.7)	3.8%	(3.3)
	12	9	1515	37.2%	(11.8)	>	28.8%	(11.3)	>	16.6%	(12.5)	>	-7.0%	(6.3)	26.3%	(5.3)
	12	12	2051	23.3%	(4.7)	>	16.4%	(6.5)	>	11.3%	(7.6)	>	-8.4%	(4.2)	20.6%	(1.9)
	12	15	2146	2.2%	(5.2)	≈	0.9%	(3.9)	>	-10.0%	(5.2)	>	-19.4%	(1.3)	7.6%	(1.8)
	15	9	1567	26.4%	(10.0)	>	17.3%	(8.2)	>	11.6%	(9.9)	>	0.9%	(3.8)	12.5%	(4.1)
	15	12	1752	35.4%	(9.0)	>	30.3%	(8.2)	>	21.5%	(9.6)	>	4.1%	(4.0)	27.0%	(2.6)
	15	15	2026	11.4%	(7.7)	>	8.1%	(4.5)	>	-1.6%	(6.3)	>	-11.0%	(1.5)	3.9%	(1.6)
instance p05	9	9	690	0.0%	(0.1)	≈	0.0%	(0.0)	≈	0.0%	(0.0)	≈	0.0%	(0.0)	13.6%	(8.2)
	9	12	888	72.8%	(20.2)	>	37.9%	(22.9)	≈	34.8%	(22.6)	>	1.2%	(4.3)	69.0%	(7.4)
	9	15	1623	59.2%	(13.7)	>	43.6%	(8.7)	>	30.8%	(12.8)	>	5.5%	(4.4)	36.2%	(4.5)
	12	9	1016	25.2%	(15.7)	>	13.7%	(12.2)	>	1.9%	(6.6)	>	-0.7%	(0.0)	23.4%	(4.5)
	12	12	1325	52.1%	(13.1)	>	31.3%	(12.1)	≈	25.3%	(22.0)	>	-11.3%	(3.8)	36.6%	(3.1)
	12	15	1986	64.9%	(11.0)	>	48.6%	(7.2)	>	37.3%	(13.1)	>	7.5%	(4.0)	34.5%	(2.6)
	15	9	1010	-15.5%	(6.9)	>	-18.9%	(0.1)	>	-19.3%	(0.1)	>	-19.4%	(0.0)	-9.4%	(1.4)
	15	12	1156	75.9%	(17.3)	>	50.8%	(13.7)	>	6.7%	(21.3)	>	-8.5%	(5.7)	51.5%	(3.9)
	15	15	1900	65.2%	(32.4)	≈	53.2%	(8.6)	>	38.1%	(15.0)	>	-3.0%	(3.7)	38.5%	(3.7)
instance p06	9	9	2184	19.9%	(7.4)	>	9.8%	(6.6)	≈	9.9%	(5.1)	>	-11.0%	(2.1)		
	9	12	2915	21.9%	(7.5)	>	8.5%	(7.0)	≈	5.5%	(7.5)	>	-13.9%	(1.8)		
	9	15	2265	61.2%	(14.2)	>	41.4%	(9.8)	>	32.5%	(17.5)	>	-1.9%	(2.9)		
	12	9	2162	33.5%	(6.6)	>	23.9%	(6.6)	≈	21.5%	(7.5)	>	-3.3%	(1.9)		
	12	12	3031	43.4%	(8.0)	>	28.1%	(4.9)	>	22.8%	(5.8)	>	-5.3%	(1.8)		
	12	15	2401	89.5%	(15.2)	>	54.8%	(11.6)	≈	47.6%	(17.2)	>	3.5%	(3.5)		
	15	9	2719	30.8%	(7.1)	>	23.2%	(4.8)	>	14.0%	(8.3)	>	-10.1%	(2.3)		
	15	12	3452	50.4%	(16.6)	>	23.7%	(5.1)	>	16.9%	(4.6)	>	-9.0%	(2.1)		
	15	15	2928	81.9%	(9.4)	>	57.2%	(10.8)	≈	56.0%	(10.6)	>	0.1%	(2.6)		
instance p07	9	9	6461	-10.9%	(4.8)	>	-16.1%	(3.1)	≈	-14.9%	(3.4)	>	-27.5%	(1.1)		
	9	12	6856	-2.0%	(4.7)	>	-5.4%	(4.2)	<	-0.7%	(6.1)	>	-25.0%	(1.6)		
	9	15	6952	11.4%	(6.7)	>	1.4%	(6.5)	<	13.1%	(7.6)	>	-26.4%	(3.9)		
	12	9	6758	-15.1%	(4.2)	>	-19.8%	(3.4)	≈	-20.6%	(3.3)	>	-34.8%	(0.9)		
	12	12	7090	2.9%	(6.7)	>	-8.5%	(6.7)	<	5.0%	(5.9)	>	-31.7%	(1.5)		
	12	15	7325	14.5%	(8.0)	>	2.6%	(7.3)	<	13.1%	(7.3)	>	-32.6%	(1.9)		
	15	9	6979	3.2%	(5.0)	>	-5.2%	(2.5)	>	-8.2%	(2.7)	>	-24.8%	(1.0)		
	15	12	7358	16.0%	(9.8)	>	-5.7%	(6.3)	<	12.8%	(6.4)	>	-26.3%	(2.5)		
	15	15	7551	26.3%	(6.2)	>	12.4%	(6.7)	<	24.7%	(6.2)	>	-25.7%	(2.4)		

				HVREA			BNREA			EBNREA			HVR+VNS		ACO	
	x	y	orig	mean	dev	p	mean	dev	p	mean	dev	p	mean	dev	mean	dev
instance p08	9	9	3467	24.7%	(5.6)	>	15.4%	(4.8)	>	12.3%	(4.5)	>	-7.4%	(1.0)		
	9	12	3978	29.2%	(6.5)	>	17.0%	(4.6)	≈	19.4%	(5.6)	>	-5.7%	(1.2)		
	9	15	3726	58.1%	(8.8)	>	34.0%	(6.2)	<	45.5%	(7.4)	>	0.5%	(1.8)		
	12	9	3901	37.3%	(7.5)	>	23.9%	(4.3)	>	21.5%	(4.7)	>	-1.5%	(1.5)		
	12	12	4305	46.5%	(8.5)	>	26.3%	(3.9)	<	34.3%	(5.3)	>	-1.7%	(1.7)		
	12	15	4225	78.5%	(8.4)	>	44.9%	(6.5)	<	56.2%	(7.2)	>	0.9%	(2.1)		
	15	9	4656	29.9%	(5.0)	>	17.9%	(3.6)	>	15.7%	(3.5)	>	-7.5%	(1.1)		
	15	12	5042	41.6%	(6.6)	>	21.6%	(5.2)	<	27.1%	(3.9)	>	-7.0%	(1.1)		
	15	15	4909	79.9%	(8.7)	>	42.7%	(9.5)	<	51.0%	(6.0)	>	-1.6%	(1.6)		
instance p09	9	9	3319	58.6%	(6.9)	>	51.5%	(5.6)	>	30.5%	(6.3)	>	9.6%	(3.2)		
	9	12	3522	58.0%	(8.3)	>	48.6%	(6.3)	>	33.8%	(4.8)	>	1.8%	(2.6)		
	9	15	4906	30.7%	(5.6)	>	21.6%	(4.8)	<	25.0%	(4.8)	>	-1.7%	(1.5)		
	12	9	3506	42.1%	(5.5)	>	36.3%	(4.9)	>	23.8%	(9.2)	>	4.3%	(3.4)		
	12	12	3706	44.3%	(7.7)	>	35.5%	(4.8)	>	27.8%	(7.0)	>	-2.0%	(3.2)		
	12	15	4922	35.1%	(9.7)	>	14.5%	(5.9)	<	26.5%	(3.9)	>	-2.0%	(2.2)		
	15	9	4460	53.4%	(4.7)	>	45.4%	(4.1)	>	30.3%	(6.0)	>	10.8%	(3.4)		
	15	12	4690	57.7%	(6.1)	>	47.3%	(4.4)	>	33.3%	(4.6)	>	3.2%	(2.6)		
	15	15	6171	42.7%	(6.0)	>	25.6%	(3.4)	≈	24.9%	(3.7)	>	-1.3%	(1.9)		
instance p10	9	9	3979	50.4%	(8.6)	>	38.6%	(7.3)	>	26.9%	(6.0)	>	6.7%	(2.8)		
	9	12	6496	15.0%	(3.1)	>	9.2%	(3.1)	≈	7.9%	(2.6)	>	-4.6%	(0.8)		
	9	15	7821	31.4%	(2.5)	>	24.6%	(1.8)	>	14.7%	(2.6)	>	-0.4%	(0.9)		
	12	9	3535	62.4%	(8.4)	>	52.2%	(8.1)	>	34.5%	(9.6)	>	9.9%	(3.3)		
	12	12	5708	24.2%	(4.4)	>	18.1%	(2.9)	>	15.6%	(3.5)	>	-0.4%	(1.2)		
	12	15	7138	36.2%	(5.3)	>	27.7%	(3.7)	>	16.7%	(3.8)	>	0.8%	(1.3)		
	15	9	5190	46.1%	(6.3)	>	38.0%	(4.9)	>	20.9%	(6.9)	>	2.0%	(2.0)		
	15	12	7183	26.8%	(4.3)	>	19.9%	(3.4)	>	11.9%	(2.6)	>	-3.6%	(1.2)		
	15	15	8356	40.2%	(3.1)	>	29.8%	(2.7)	>	20.2%	(1.9)	>	0.6%	(1.6)		

Table A.2: The mean percentage gaps over 30 runs and corresponding standard deviations for the tested configurations **with** the application of VNS in the end are listed here. For comparisons the available ACO results from [29] are also presented. Moreover the values in columns p correspond to the student t-test with 5% error level and indicate the relation between two neighbouring configurations.

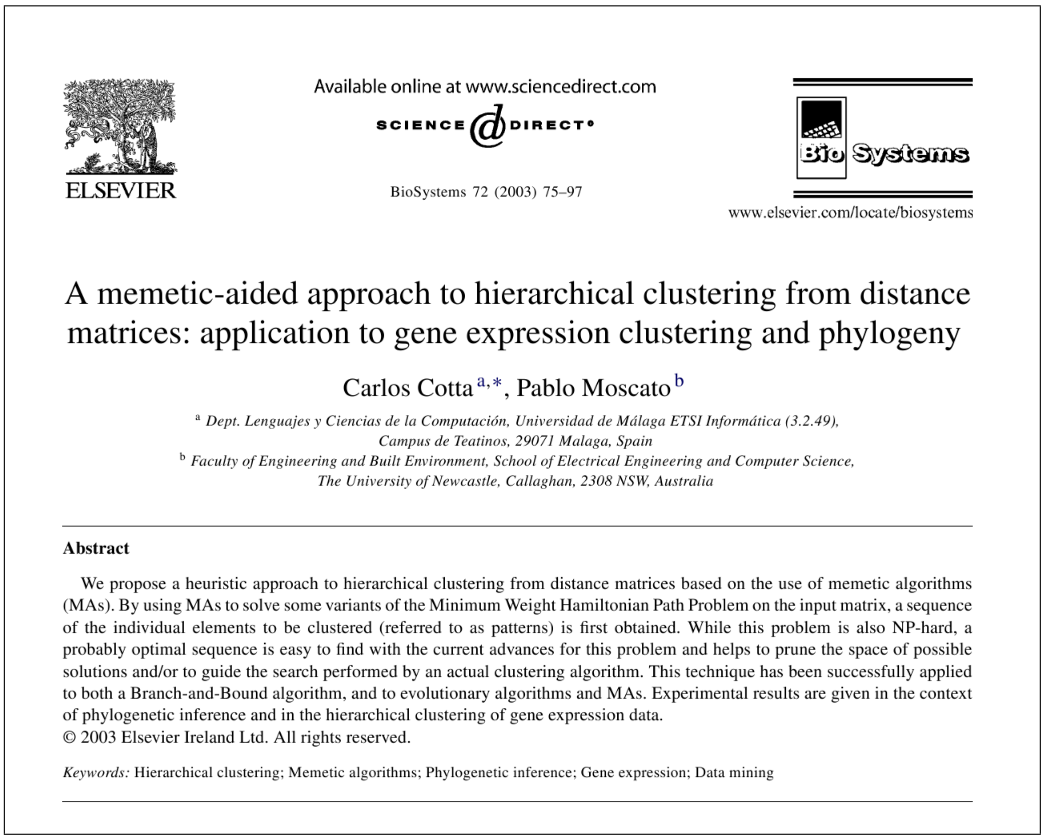
				HVREA			BNREA			EBNREA			HVR+VNS		ACO	
	x	y	orig	mean	dev	p	mean	dev	p	mean	dev	p	mean	dev	mean	dev
instance p01	9	9	2094	0,0%	(0,0)	≈	0,0%	(0,0)	≈	0,0%	(0,0)	≈	0,0%	(0,0)	0,0%	(0,0)
	9	12	3142	27,6%	(5,3)	≈	28,8%	(6,1)	>	21,1%	(4,9)	>	6,9%	(5,7)	21,0%	(4,8)
	9	15	3223	30,7%	(6,2)	≈	28,4%	(5,8)	>	20,1%	(10,3)	>	6,3%	(6,3)	30,7%	(2,1)
	12	9	2907	32,5%	(6,5)	>	19,4%	(6,8)	≈	16,6%	(9,8)	>	0,5%	(7,6)	25,2%	(3,2)
	12	12	3695	26,2%	(4,4)	≈	23,6%	(5,6)	>	20,7%	(3,4)	>	6,6%	(2,3)	27,5%	(2,9)
	12	15	3825	28,7%	(6,3)	≈	28,1%	(3,6)	>	24,2%	(4,3)	>	10,2%	(2,0)	32,7%	(3,2)
	15	9	2931	31,6%	(5,9)	>	28,4%	(4,5)	>	1,6%	(6,2)	≈	0,0%	(0,0)	29,1%	(4,1)
	15	12	3732	29,5%	(5,3)	≈	28,6%	(5,7)	>	22,4%	(7,7)	>	8,9%	(4,3)	32,5%	(2,6)
	15	15	3870	32,2%	(4,5)	≈	30,1%	(5,9)	≈	28,4%	(3,3)	>	11,5%	(2,2)	33,2%	(3,2)

				HVREA			BNREA			EBNREA			HVR+VNS		ACO	
	x	y	orig	mean	dev	p	mean	dev	p	mean	dev	p	mean	dev	mean	dev
instance p02	9	9	1434	-9,2%	(7,7)	>	-13,3%	(7,4)	>	-24,0%	(5,8)	>	-29,7%	(0,6)	4,8%	(4,6)
	9	12	1060	26,4%	(10,9)	≈	28,7%	(9,4)	>	6,1%	(6,1)	>	0,0%	(0,4)	35,3%	(5,1)
	9	15	1978	6,2%	(5,0)	>	2,3%	(3,8)	>	-0,7%	(3,2)	>	-11,0%	(0,6)	21,7%	(3,6)
	12	9	1396	-10,6%	(7,0)	>	-14,0%	(4,6)	>	-18,7%	(8,6)	>	-30,4%	(1,4)	13,4%	(4,2)
	12	12	1083	21,9%	(9,8)	>	9,4%	(5,2)	>	5,2%	(8,9)	>	-0,3%	(1,1)	27,7%	(5,0)
	12	15	1904	5,2%	(5,0)	>	-5,1%	(3,4)	<	0,8%	(6,1)	>	-10,8%	(2,0)	19,7%	(3,5)
	15	9	1658	7,2%	(5,6)	≈	6,6%	(7,0)	>	-7,2%	(7,8)	>	-16,3%	(2,0)	13,8%	(4,0)
	15	12	1503	17,7%	(9,5)	≈	13,2%	(8,3)	>	7,4%	(8,7)	>	0,1%	(0,5)	19,8%	(3,3)
	15	15	2283	5,6%	(4,1)	≈	4,0%	(4,2)	≈	2,2%	(4,7)	>	-6,3%	(2,3)	12,6%	(1,7)
instance p03	9	9	2486	5,8%	(6,0)	>	-3,2%	(4,0)	>	-5,8%	(4,4)	≈	-5,8%	(5,0)	6,7%	(4,8)
	9	12	2651	26,8%	(6,6)	>	22,6%	(6,4)	>	11,8%	(8,1)	>	0,9%	(3,7)	30,0%	(3,6)
	9	15	2551	18,3%	(6,9)	>	12,4%	(5,5)	>	2,1%	(3,8)	>	-0,3%	(1,3)	18,0%	(2,6)
	12	9	3075	15,3%	(5,1)	>	11,9%	(5,5)	≈	9,9%	(4,8)	>	4,0%	(1,2)	11,6%	(2,9)
	12	12	3377	22,2%	(6,6)	≈	21,4%	(5,2)	>	16,3%	(5,2)	>	0,5%	(4,7)	23,3%	(2,5)
	12	15	3313	16,8%	(6,3)	>	13,2%	(6,9)	>	2,3%	(4,7)	>	-3,5%	(0,4)	16,3%	(2,1)
	15	9	3213	19,7%	(5,1)	>	16,1%	(4,7)	>	11,7%	(5,2)	>	0,1%	(2,5)	13,6%	(2,3)
	15	12	3278	31,0%	(6,3)	≈	28,4%	(7,2)	>	22,6%	(11,1)	>	10,7%	(4,3)	36,3%	(2,1)
	15	15	3308	24,3%	(4,7)	>	21,7%	(5,0)	>	5,7%	(6,3)	>	0,3%	(1,9)	27,0%	(1,4)
instance p04	9	9	1104	14,5%	(13,8)	>	3,1%	(15,0)	>	-13,7%	(14,6)	>	-25,0%	(10,8)	18,4%	(7,2)
	9	12	1463	11,2%	(10,1)	>	5,5%	(8,3)	>	-3,2%	(9,7)	>	-15,2%	(2,9)	15,6%	(4,5)
	9	15	1589	-4,2%	(5,4)	>	-7,8%	(5,0)	>	-14,5%	(8,4)	>	-24,0%	(1,7)	3,8%	(3,3)
	12	9	1515	28,3%	(11,4)	≈	24,0%	(10,9)	>	14,9%	(10,9)	>	-7,1%	(6,3)	26,3%	(5,3)
	12	12	2051	17,4%	(3,3)	>	13,4%	(4,7)	>	9,5%	(6,4)	>	-8,4%	(4,2)	20,6%	(1,9)
	12	15	2146	-1,8%	(3,6)	≈	-0,8%	(3,8)	>	-10,6%	(5,0)	>	-19,4%	(1,3)	7,6%	(1,8)
	15	9	1567	20,9%	(7,9)	>	14,7%	(8,1)	≈	11,2%	(9,5)	>	0,9%	(3,8)	12,5%	(4,1)
	15	12	1752	29,9%	(7,6)	≈	27,1%	(7,7)	>	20,8%	(9,2)	>	4,0%	(4,0)	27,0%	(2,6)
	15	15	2026	5,1%	(5,1)	≈	3,2%	(4,2)	>	-2,7%	(6,0)	>	-11,0%	(1,5)	3,9%	(1,6)
instance p05	9	9	690	0,0%	(0,1)	≈	0,0%	(0,0)	≈	0,0%	(0,0)	≈	0,0%	(0,0)	13,6%	(8,2)
	9	12	888	61,7%	(19,9)	>	33,8%	(23,2)	≈	33,1%	(21,1)	>	1,2%	(4,3)	69,0%	(7,4)
	9	15	1623	41,9%	(9,8)	>	35,5%	(8,7)	>	27,5%	(11,9)	>	5,4%	(4,4)	36,2%	(4,5)
	12	9	1016	21,6%	(14,9)	>	12,2%	(11,1)	>	1,7%	(6,2)	>	-0,7%	(0,0)	23,4%	(4,5)
	12	12	1325	35,8%	(10,9)	>	24,1%	(10,8)	≈	19,9%	(18,0)	>	-11,3%	(3,8)	36,6%	(3,1)
	12	15	1986	37,2%	(5,0)	≈	35,9%	(5,4)	≈	31,7%	(11,0)	>	7,4%	(4,0)	34,5%	(2,6)
	15	9	1010	-17,0%	(5,5)	≈	-19,0%	(0,2)	>	-19,3%	(0,1)	>	-19,4%	(0,0)	-9,4%	(1,4)
	15	12	1156	56,8%	(12,6)	>	38,7%	(14,9)	>	3,9%	(17,7)	>	-8,5%	(5,7)	51,5%	(3,9)
	15	15	1900	28,4%	(25,6)	≈	31,0%	(6,3)	>	24,2%	(11,0)	>	-3,2%	(3,6)	38,5%	(3,7)
instance p06	9	9	2184	11,4%	(5,2)	>	4,8%	(5,3)	≈	6,7%	(4,9)	>	-11,1%	(2,0)		
	9	12	2915	6,6%	(3,6)	>	1,8%	(5,0)	≈	0,4%	(5,6)	>	-13,9%	(1,8)		
	9	15	2265	29,5%	(5,2)	≈	27,0%	(7,0)	>	20,1%	(10,8)	>	-2,0%	(3,0)		
	12	9	2162	21,5%	(5,2)	>	15,6%	(5,0)	≈	13,6%	(4,4)	>	-3,3%	(1,8)		
	12	12	3031	17,1%	(3,7)	>	13,8%	(3,3)	≈	12,1%	(4,2)	>	-5,3%	(1,7)		
	12	15	2401	44,6%	(7,3)	>	39,4%	(7,8)	>	29,3%	(8,8)	>	3,4%	(3,5)		
	15	9	2719	12,8%	(4,3)	≈	12,1%	(3,7)	>	4,3%	(5,5)	>	-10,3%	(2,3)		
	15	12	3452	10,9%	(3,9)	>	8,1%	(3,0)	>	4,9%	(3,5)	>	-9,1%	(2,1)		
	15	15	2928	33,0%	(5,5)	≈	30,3%	(4,9)	>	22,8%	(5,8)	>	0,0%	(2,6)		

				HVREA			BNREA			EBNREA			HVR+VNS		ACO	
	x	y	orig	mean	dev	p	mean	dev	p	mean	dev	p	mean	dev	mean	dev
instance p07	9	9	6461	-19,9%	(3,4)	≈	-20,8%	(2,3)	≈	-20,5%	(2,8)	>	-27,5%	(1,1)		
	9	12	6856	-11,3%	(3,8)	≈	-12,6%	(3,6)	≈	-12,6%	(4,5)	>	-25,0%	(1,6)		
	9	15	6952	-10,3%	(3,5)	≈	-10,0%	(4,2)	≈	-8,0%	(6,1)	>	-26,4%	(3,9)		
	12	9	6758	-26,3%	(3,1)	≈	-26,9%	(3,0)	>	-28,5%	(2,4)	>	-34,8%	(0,9)		
	12	12	7090	-16,9%	(3,5)	>	-19,9%	(4,3)	≈	-19,8%	(4,3)	>	-31,8%	(1,5)		
	12	15	7325	-15,5%	(3,2)	≈	-14,1%	(4,6)	>	-17,4%	(3,9)	>	-32,7%	(1,9)		
	15	9	6979	-15,2%	(2,3)	≈	-15,5%	(2,4)	>	-17,5%	(1,7)	>	-24,8%	(1,0)		
	15	12	7358	-11,7%	(3,6)	>	-16,0%	(5,2)	<	-11,9%	(6,2)	>	-26,4%	(2,6)		
	15	15	7551	-10,5%	(3,2)	≈	-9,6%	(4,0)	≈	-9,4%	(3,6)	>	-25,7%	(2,4)		
instance p08	9	9	3467	9,1%	(3,8)	>	5,1%	(3,5)	≈	6,3%	(3,0)	>	-7,5%	(1,1)		
	9	12	3978	10,4%	(3,4)	>	7,9%	(3,2)	≈	8,1%	(3,1)	>	-5,8%	(1,2)		
	9	15	3726	20,2%	(3,9)	≈	18,3%	(4,8)	≈	19,8%	(4,3)	>	0,4%	(1,8)		
	12	9	3901	17,6%	(3,8)	>	13,6%	(5,0)	≈	13,2%	(2,6)	>	-1,6%	(1,5)		
	12	12	4305	16,6%	(3,7)	>	13,2%	(3,0)	≈	13,5%	(3,1)	>	-1,8%	(1,7)		
	12	15	4225	23,3%	(3,4)	>	21,2%	(4,4)	≈	22,4%	(5,1)	>	0,8%	(2,1)		
	15	9	4656	7,2%	(4,0)	>	3,3%	(3,2)	≈	3,9%	(2,4)	>	-7,6%	(1,1)		
	15	12	5042	7,3%	(2,8)	>	5,1%	(2,3)	≈	5,3%	(2,3)	>	-7,1%	(1,1)		
	15	15	4909	17,2%	(4,0)	>	14,9%	(3,7)	≈	14,3%	(2,6)	>	-1,7%	(1,6)		
instance p09	9	9	3319	42,1%	(7,2)	≈	39,3%	(5,7)	>	27,7%	(6,1)	>	9,6%	(3,2)		
	9	12	3522	32,5%	(6,7)	≈	31,3%	(6,1)	>	23,9%	(4,8)	>	1,7%	(2,6)		
	9	15	4906	15,6%	(4,4)	>	12,5%	(3,6)	≈	11,9%	(4,4)	>	-1,7%	(1,5)		
	12	9	3506	29,1%	(4,1)	≈	27,1%	(5,5)	>	20,9%	(8,4)	>	4,2%	(3,4)		
	12	12	3706	23,1%	(5,4)	≈	21,4%	(4,2)	≈	19,3%	(6,1)	>	-2,1%	(3,2)		
	12	15	4922	12,3%	(3,7)	>	7,9%	(3,9)	<	14,5%	(3,9)	>	-2,1%	(2,2)		
	15	9	4460	34,7%	(4,6)	≈	34,9%	(4,2)	>	25,3%	(4,9)	>	10,7%	(3,4)		
	15	12	4690	27,0%	(4,1)	≈	27,9%	(3,8)	>	20,4%	(3,2)	>	3,1%	(2,7)		
	15	15	6171	13,4%	(3,0)	≈	13,3%	(3,3)	>	11,6%	(2,3)	>	-1,3%	(1,9)		
instance p10	9	9	3979	28,1%	(6,7)	≈	24,8%	(7,4)	>	20,4%	(4,9)	>	6,7%	(2,8)		
	9	12	6496	5,2%	(2,4)	>	3,7%	(2,3)	≈	3,3%	(2,0)	>	-4,6%	(0,8)		
	9	15	7821	10,3%	(2,1)	≈	10,8%	(2,4)	>	6,9%	(2,0)	>	-0,4%	(0,9)		
	12	9	3535	34,5%	(6,9)	≈	32,1%	(5,4)	>	23,7%	(6,9)	>	9,8%	(3,3)		
	12	12	5708	12,3%	(2,8)	>	10,5%	(2,6)	≈	10,1%	(2,8)	>	-0,5%	(1,3)		
	12	15	7138	13,5%	(2,8)	≈	12,6%	(2,2)	>	10,3%	(2,9)	>	0,7%	(1,3)		
	15	9	5190	23,5%	(4,5)	>	20,0%	(4,5)	>	12,5%	(3,3)	>	2,0%	(2,0)		
	15	12	7183	10,4%	(3,0)	>	8,0%	(2,8)	>	5,3%	(2,1)	>	-3,6%	(1,2)		
	15	15	8356	12,1%	(2,5)	≈	12,2%	(2,3)	>	9,4%	(2,1)	>	0,6%	(1,6)		

APPENDIX B

INSTANCES



Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

BioSystems 72 (2003) 75–97

BioSystems

www.elsevier.com/locate/biosystems

A memetic-aided approach to hierarchical clustering from distance matrices: application to gene expression clustering and phylogeny

Carlos Cotta[a,*], Pablo Moscato[b]

[a] *Dept. Lenguajes y Ciencias de la Computación, Universidad de Málaga ETSI Informática (3.2.49), Campus de Teatinos, 29071 Malaga, Spain*

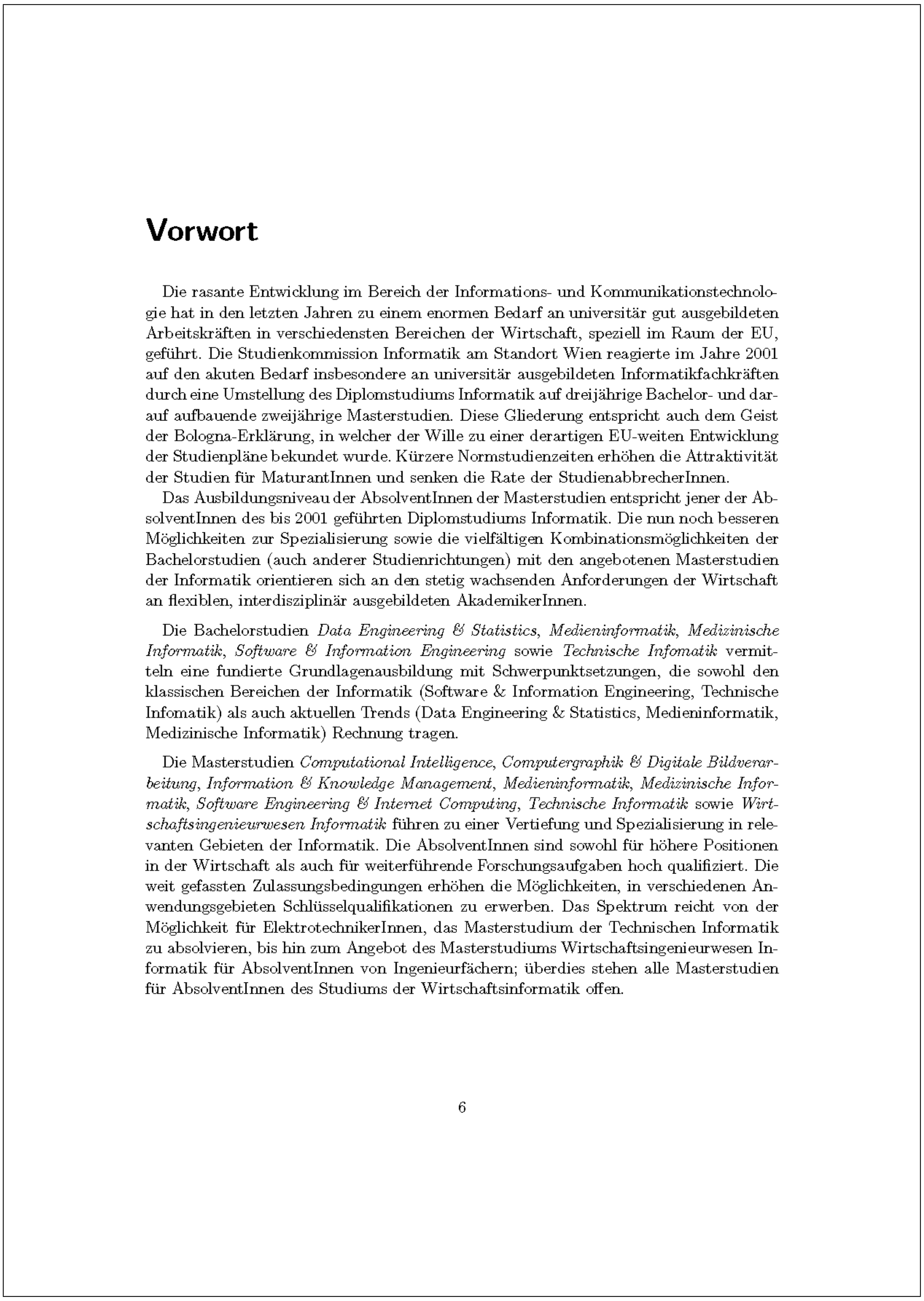
[b] *Faculty of Engineering and Built Environment, School of Electrical Engineering and Computer Science, The University of Newcastle, Callaghan, 2308 NSW, Australia*

Abstract

We propose a heuristic approach to hierarchical clustering from distance matrices based on the use of memetic algorithms (MAs). By using MAs to solve some variants of the Minimum Weight Hamiltonian Path Problem on the input matrix, a sequence of the individual elements to be clustered (referred to as patterns) is first obtained. While this problem is also NP-hard, a probably optimal sequence is easy to find with the current advances for this problem and helps to prune the space of possible solutions and/or to guide the search performed by an actual clustering algorithm. This technique has been successfully applied to both a Branch-and-Bound algorithm, and to evolutionary algorithms and MAs. Experimental results are given in the context of phylogenetic inference and in the hierarchical clustering of gene expression data.
© 2003 Elsevier Ireland Ltd. All rights reserved.

Keywords: Hierarchical clustering; Memetic algorithms; Phylogenetic inference; Gene expression; Data mining

Figure B.1: Instance p06



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.

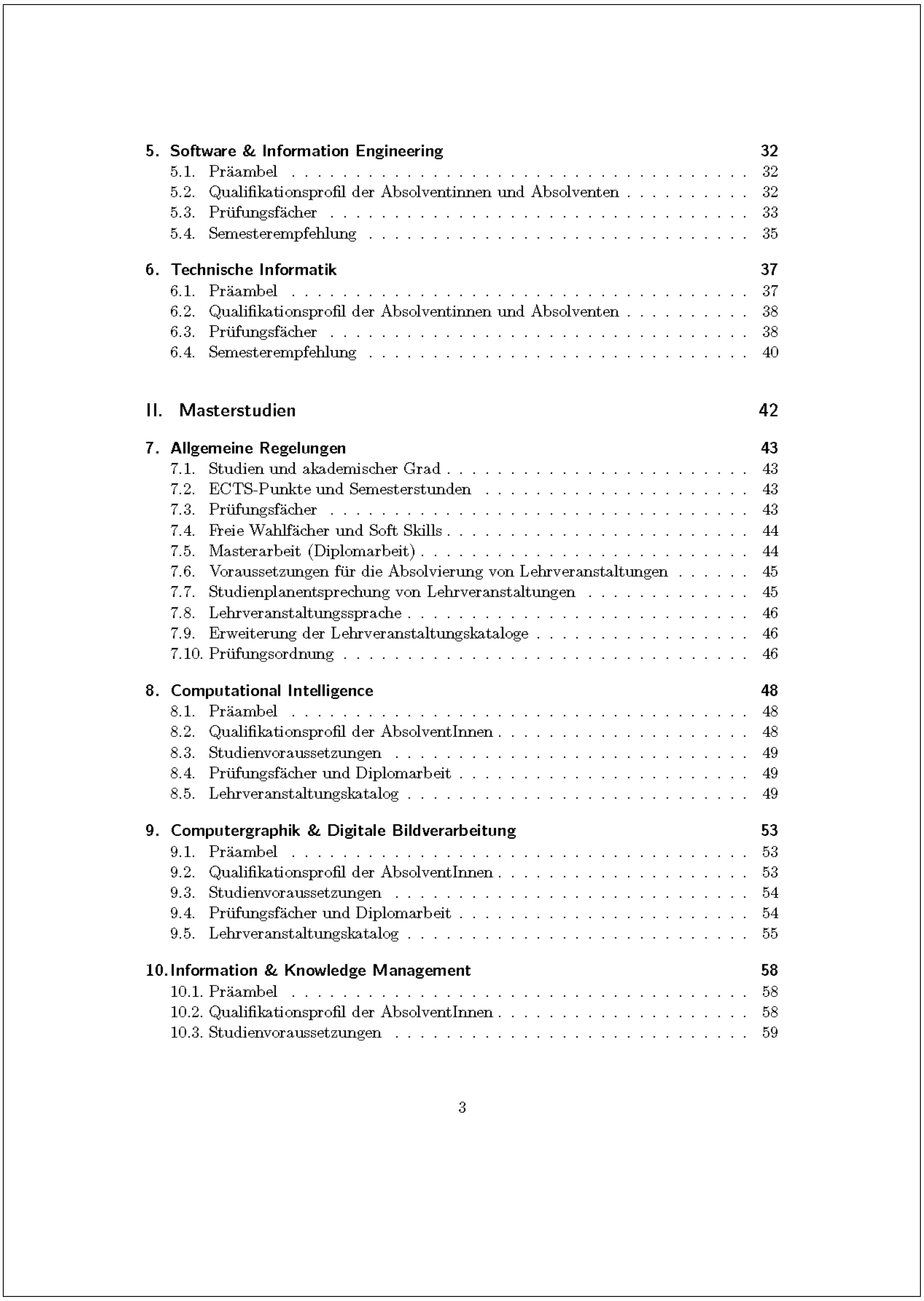
Das Ausbildungsniveau der AbsolventInnen der Masterstudien entspricht jener der AbsolventInnen des bis 2001 geführten Diplomstudiums Informatik. Die nun noch besseren Möglichkeiten zur Spezialisierung sowie die vielfältigen Kombinationsmöglichkeiten der Bachelorstudien (auch anderer Studienrichtungen) 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.

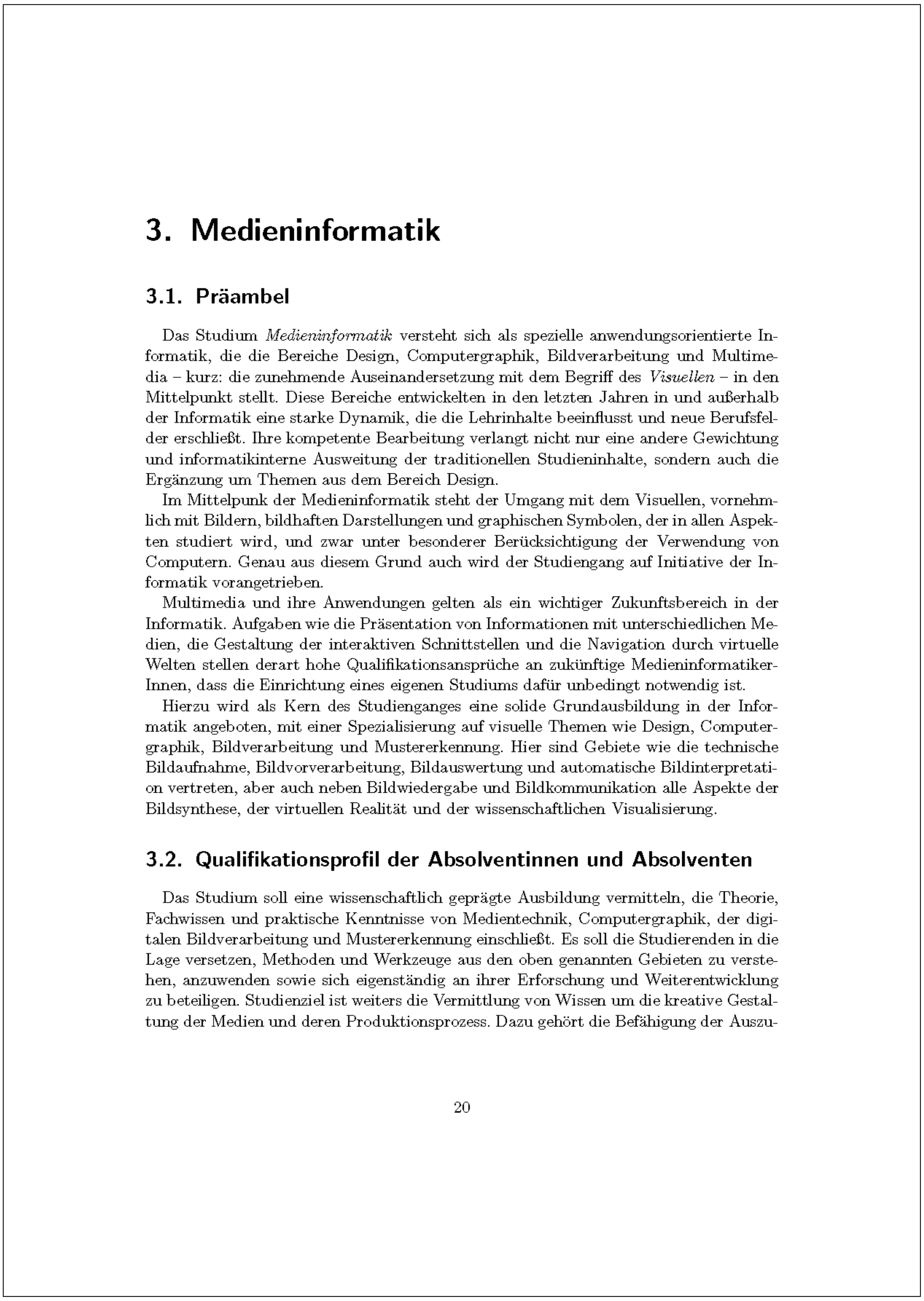
6

Figure B.2: Instance p01



3

Figure B.3: Instance p02



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.

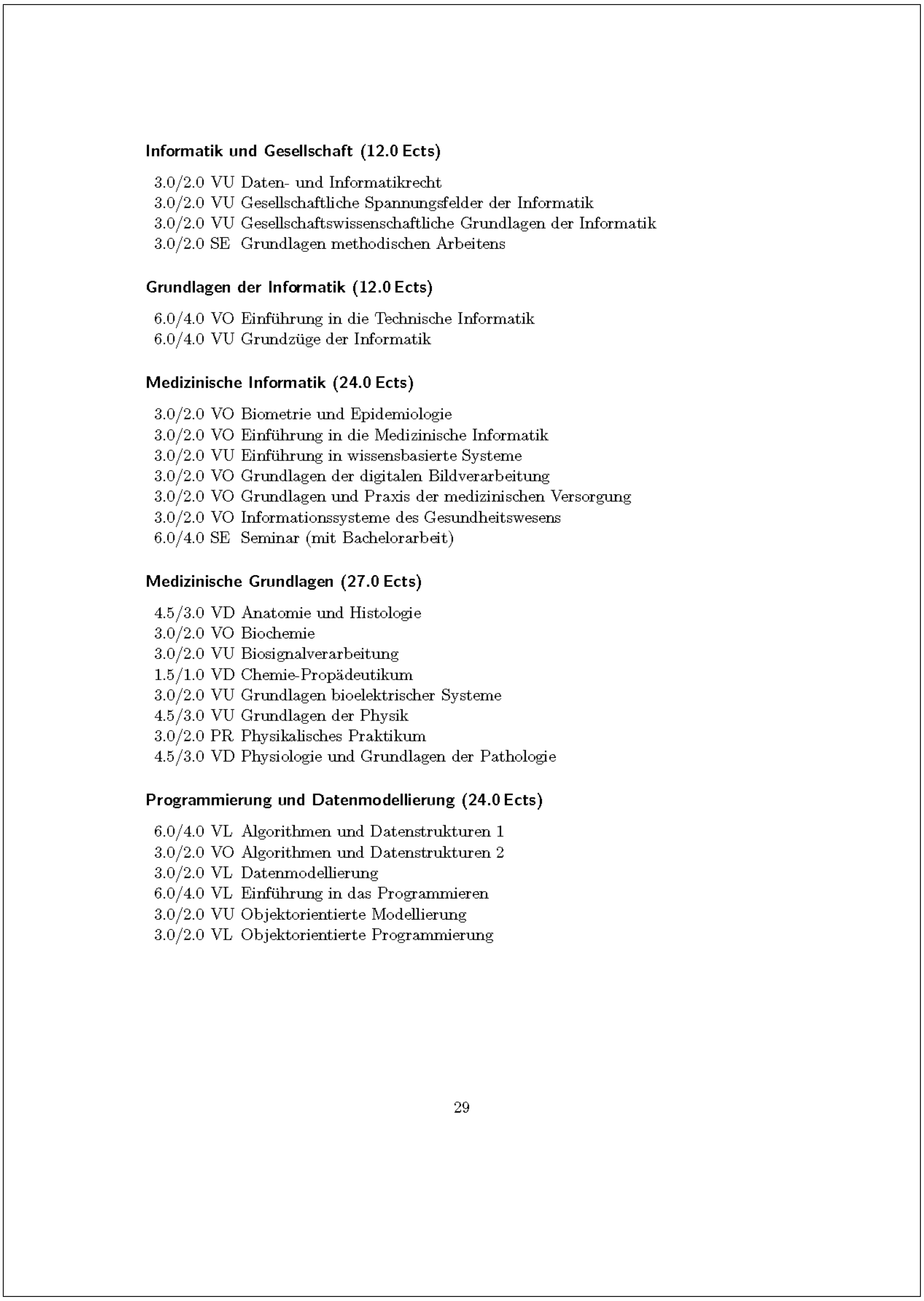
Hierzu wird als Kern des Studienganges eine solide Grundausbildung in der Informatik angeboten, mit einer Spezialisierung auf visuelle Themen wie Design, Computergraphik, Bildverarbeitung und Mustererkennung. Hier sind Gebiete wie die technische Bildaufnahme, Bildvorverarbeitung, Bildauswertung und automatische 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 B.4: Instance p03



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)

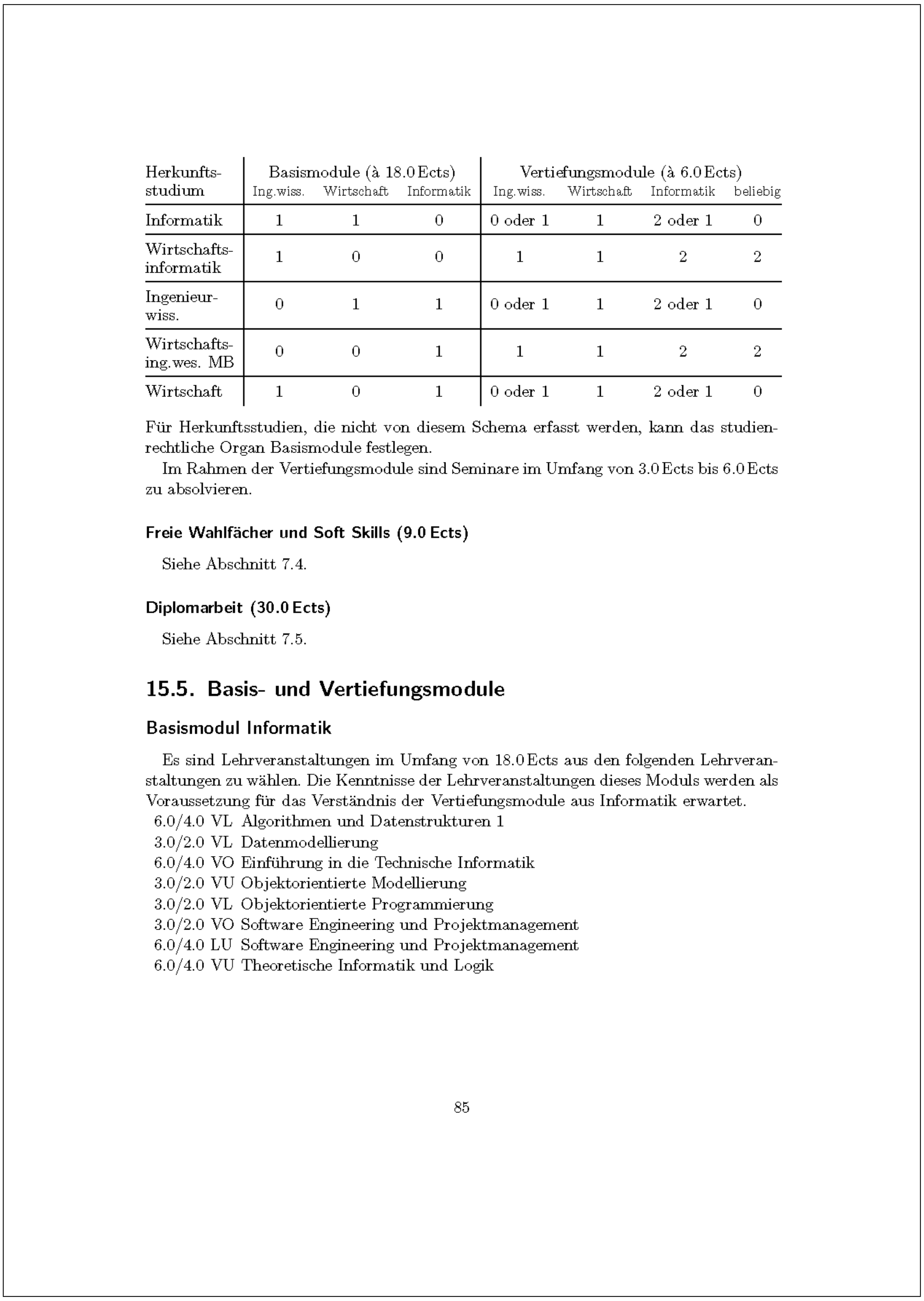
4.5/3.0	VD	Anatomie und Histologie
3.0/2.0	VO	Biochemie
3.0/2.0	VU	Biosignalverarbeitung
1.5/1.0	VD	Chemie-Propädeutikum
3.0/2.0	VU	Grundlagen bioelektrischer Systeme
4.5/3.0	VU	Grundlagen der Physik
3.0/2.0	PR	Physikalisches Praktikum
4.5/3.0	VD	Physiologie und Grundlagen der Pathologie

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 B.5: Instance p04



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

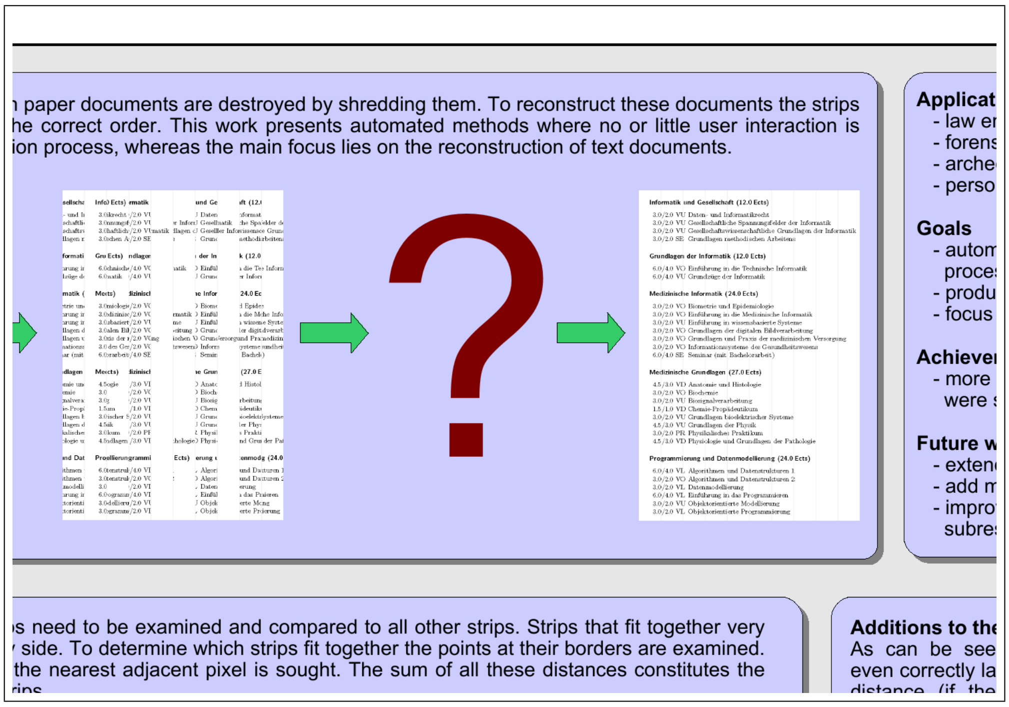
Basismodul Informatik

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 B.6: Instance p05



ı paper documents are destroyed by shredding them. To reconstruct these documents the strips
he correct order. This work presents automated methods where no or little user interaction is
ion process, whereas the main focus lies on the reconstruction of text documents.

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

Applicat
- law er
- forens
- arche
- perso

Goals
- autom
proce
- produ
- focus

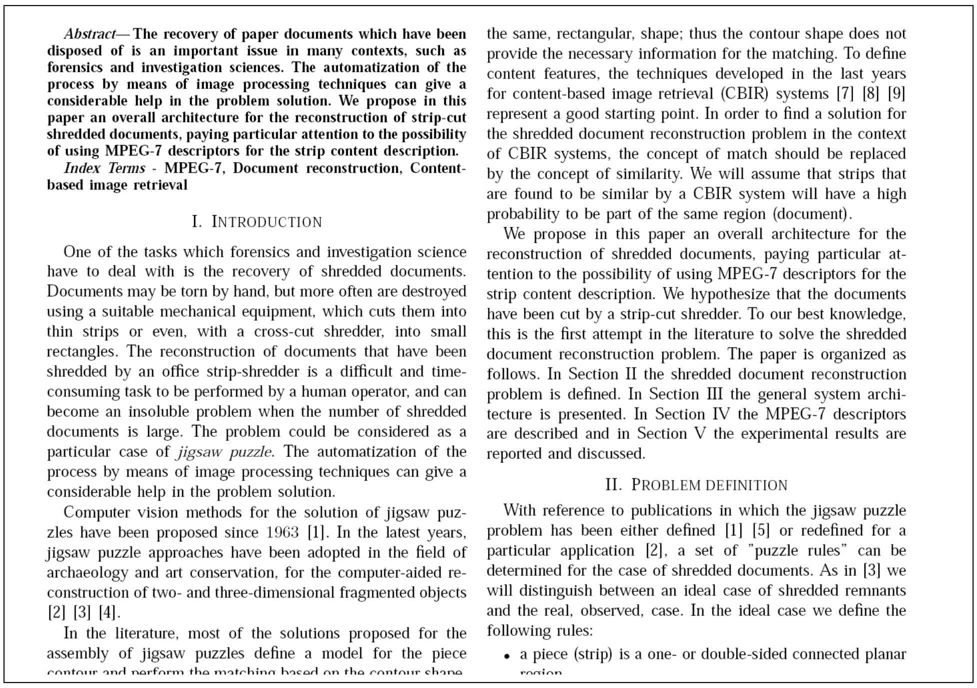
Achieve
- more
were

Future w
- exten
- add m
- impro
subre

ıs need to be examined and compared to all other strips. Strips that fit together very
side. To determine which strips fit together the points at their borders are examined.
the nearest adjacent pixel is sought. The sum of all these distances constitutes the

Additions to the
As can be see
even correctly la

Figure B.7: Instance p07



Abstract— **The recovery of paper documents which have been disposed of is an important issue in many contexts, such as forensics and investigation sciences. The automatization of the process by means of image processing techniques can give a considerable help in the problem solution. We propose in this paper an overall architecture for the reconstruction of strip-cut shredded documents, paying particular attention to the possibility of using MPEG-7 descriptors for the strip content description.**

Index Terms - **MPEG-7, Document reconstruction, Content-based image retrieval**

I. Introduction

One of the tasks which forensics and investigation science have to deal with is the recovery of shredded documents. Documents may be torn by hand, but more often are destroyed using a suitable mechanical equipment, which cuts them into thin strips or even, with a cross-cut shredder, into small rectangles. The reconstruction of documents that have been shredded by an office strip-shredder is a difficult and time-consuming task to be performed by a human operator, and can become an insoluble problem when the number of shredded documents is large. The problem could be considered as a particular case of *jigsaw puzzle*. The automatization of the process by means of image processing techniques can give a considerable help in the problem solution.

Computer vision methods for the solution of jigsaw puzzles have been proposed since 1963 [1]. In the latest years, jigsaw puzzle approaches have been adopted in the field of archaeology and art conservation, for the computer-aided reconstruction of two- and three-dimensional fragmented objects [2] [3] [4].

In the literature, most of the solutions proposed for the assembly of jigsaw puzzles define a model for the piece contour and perform the matching based on the contour shape. the same, rectangular, shape; thus the contour shape does not provide the necessary information for the matching. To define content features, the techniques developed in the last years for content-based image retrieval (CBIR) systems [7] [8] [9] represent a good starting point. In order to find a solution for the shredded document reconstruction problem in the context of CBIR systems, the concept of match should be replaced by the concept of similarity. We will assume that strips that are found to be similar by a CBIR system will have a high probability to be part of the same region (document).

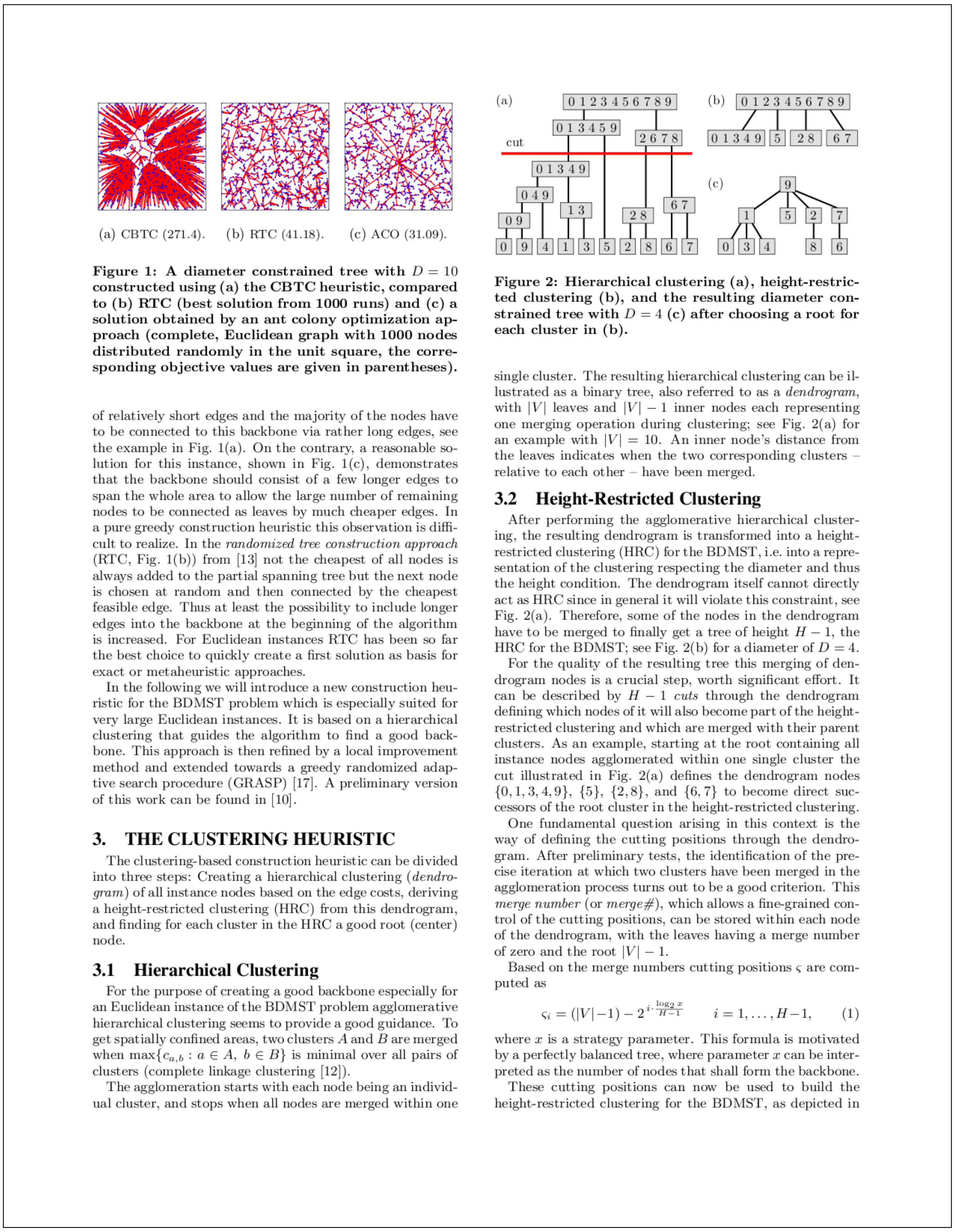
We propose in this paper an overall architecture for the reconstruction of shredded documents, paying particular attention to the possibility of using MPEG-7 descriptors for the strip content description. We hypothesize that the documents have been cut by a strip-cut shredder. To our best knowledge, this is the first attempt in the literature to solve the shredded document reconstruction problem. The paper is organized as follows. In Section II the shredded document reconstruction problem is defined. In Section III the general system architecture is presented. In Section IV the MPEG-7 descriptors are described and in Section V the experimental results are reported and discussed.

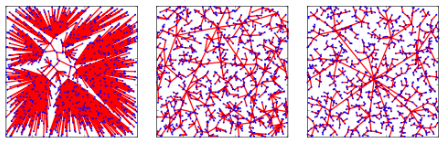
II. Problem Definition

With reference to publications in which the jigsaw puzzle problem has been either defined [1] [5] or redefined for a particular application [2], a set of "puzzle rules" can be determined for the case of shredded documents. As in [3] we will distinguish between an ideal case of shredded remnants and the real, observed, case. In the ideal case we define the following rules:

- a piece (strip) is a one- or double-sided connected planar

Figure B.8: Instance p08





(a) CBTC (271.4). (b) RTC (41.18). (c) ACO (31.09).

Figure 1: A diameter constrained tree with $D = 10$ constructed using (a) the CBTC heuristic, compared to (b) RTC (best solution from 1000 runs) and (c) a solution obtained by an ant colony optimization approach (complete, Euclidean graph with 1000 nodes distributed randomly in the unit square, the corresponding objective values are given in parentheses).

of relatively short edges and the majority of the nodes have to be connected to this backbone via rather long edges, see the example in Fig. 1(a). On the contrary, a reasonable solution for this instance, shown in Fig. 1(c), demonstrates that the backbone should consist of a few longer edges to span the whole area to allow the large number of remaining nodes to be connected as leaves by much cheaper edges. In a pure greedy construction heuristic this observation is difficult to realize. In the *randomized tree construction approach* (RTC, Fig. 1(b)) from [13] not the cheapest of all nodes is always added to the partial spanning tree but the next node is chosen at random and then connected by the cheapest feasible edge. Thus at least the possibility to include longer edges into the backbone at the beginning of the algorithm is increased. For Euclidean instances RTC has been so far the best choice to quickly create a first solution as basis for exact or metaheuristic approaches.

In the following we will introduce a new construction heuristic for the BDMST problem which is especially suited for very large Euclidean instances. It is based on a hierarchical clustering that guides the algorithm to find a good backbone. This approach is then refined by a local improvement method and extended towards a greedy randomized adaptive search procedure (GRASP) [17]. A preliminary version of this work can be found in [10].

3. THE CLUSTERING HEURISTIC

The clustering-based construction heuristic can be divided into three steps: Creating a hierarchical clustering (*dendrogram*) of all instance nodes based on the edge costs, deriving a height-restricted clustering (HRC) from this dendrogram, and finding for each cluster in the HRC a good root (center) node.

3.1 Hierarchical Clustering

For the purpose of creating a good backbone especially for an Euclidean instance of the BDMST problem agglomerative hierarchical clustering seems to provide a good guidance. To get spatially confined areas, two clusters A and B are merged when $\max\{c_{a,b} : a \in A,\ b \in B\}$ is minimal over all pairs of clusters (complete linkage clustering [12]).

The agglomeration starts with each node being an individual cluster, and stops when all nodes are merged within one

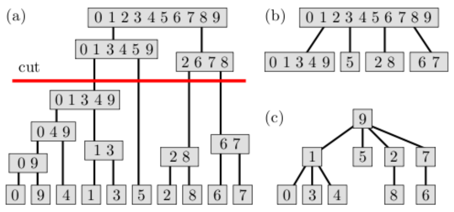


Figure 2: Hierarchical clustering (a), height-restricted clustering (b), and the resulting diameter constrained tree with $D = 4$ (c) after choosing a root for each cluster in (b).

single cluster. The resulting hierarchical clustering can be illustrated as a binary tree, also referred to as a *dendrogram*, with $|V|$ leaves and $|V| - 1$ inner nodes each representing one merging operation during clustering; see Fig. 2(a) for an example with $|V| = 10$. An inner node's distance from the leaves indicates when the two corresponding clusters – relative to each other – have been merged.

3.2 Height-Restricted Clustering

After performing the agglomerative hierarchical clustering, the resulting dendrogram is transformed into a height-restricted clustering (HRC) for the BDMST, i.e. into a representation of the clustering respecting the diameter and thus the height condition. The dendrogram itself cannot directly act as HRC since in general it will violate this constraint, see Fig. 2(a). Therefore, some of the nodes in the dendrogram have to be merged to finally get a tree of height $H - 1$, the HRC for the BDMST; see Fig. 2(b) for a diameter of $D = 4$.

For the quality of the resulting tree this merging of dendrogram nodes is a crucial step, worth significant effort. It can be described by $H - 1$ *cuts* through the dendrogram defining which nodes of it will also become part of the height-restricted clustering and which are merged with their parent clusters. As an example, starting at the root containing all instance nodes agglomerated within one single cluster the cut illustrated in Fig. 2(a) defines the dendrogram nodes $\{0, 1, 3, 4, 9\}$, $\{5\}$, $\{2, 8\}$, and $\{6, 7\}$ to become direct successors of the root cluster in the height-restricted clustering.

One fundamental question arising in this context is the way of defining the cutting positions through the dendrogram. After preliminary tests, the identification of the precise iteration at which two clusters have been merged in the agglomeration process turns out to be a good criterion. This *merge number* (or *merge#*), which allows a fine-grained control of the cutting positions, can be stored within each node of the dendrogram, with the leaves having a merge number of zero and the root $|V| - 1$.

Based on the merge numbers cutting positions ς are computed as

$$\varsigma_i = (|V|-1) - 2^{\,i \cdot \frac{\log_2 x}{H-1}} \qquad i = 1, \ldots, H-1, \tag{1}$$

where x is a strategy parameter. This formula is motivated by a perfectly balanced tree, where parameter x can be interpreted as the number of nodes that shall form the backbone.

These cutting positions can now be used to build the height-restricted clustering for the BDMST, as depicted in

Figure B.9: Instance p09

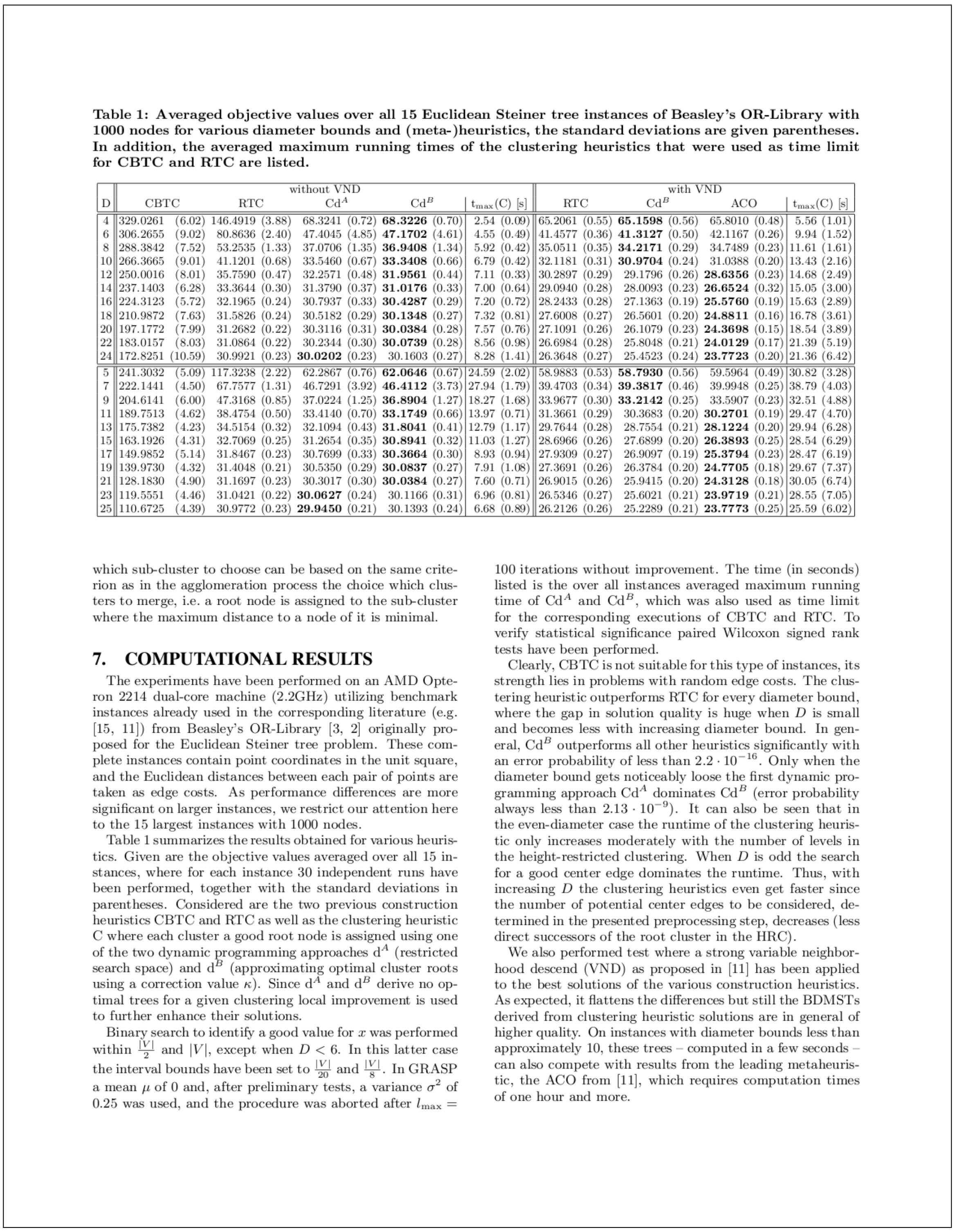


Table 1: Averaged objective values over all 15 Euclidean Steiner tree instances of Beasley's OR-Library with 1000 nodes for various diameter bounds and (meta-)heuristics, the standard deviations are given parentheses. In addition, the averaged maximum running times of the clustering heuristics that were used as time limit for CBTC and RTC are listed.

	without VND					with VND			
D	CBTC	RTC	Cd^A	Cd^B	t_{max}(C) [s]	RTC	Cd^B	ACO	t_{max}(C) [s]
4	329.0261 (6.02)	146.4919 (3.88)	68.3241 (0.72)	**68.3226** (0.70)	2.54 (0.09)	65.2061 (0.55)	**65.1598** (0.56)	65.8010 (0.48)	5.56 (1.01)
6	306.2655 (9.02)	80.8636 (2.40)	47.4045 (4.85)	**47.1702** (4.61)	4.55 (0.49)	41.4577 (0.36)	**41.3127** (0.50)	42.1167 (0.26)	9.94 (1.52)
8	288.3842 (7.52)	53.2535 (1.33)	37.0706 (1.35)	**36.9408** (1.34)	5.92 (0.42)	35.0511 (0.35)	**34.2171** (0.29)	34.7489 (0.23)	11.61 (1.61)
10	266.3665 (9.01)	41.1201 (0.68)	33.5460 (0.67)	**33.3408** (0.66)	6.79 (0.42)	32.1181 (0.31)	**30.9704** (0.24)	31.0388 (0.20)	13.43 (2.16)
12	250.0016 (8.01)	35.7590 (0.47)	32.2571 (0.48)	**31.9561** (0.44)	7.11 (0.33)	30.2897 (0.29)	29.1796 (0.26)	**28.6356** (0.23)	14.68 (2.49)
14	237.1403 (6.28)	33.3644 (0.30)	31.3790 (0.37)	**31.0176** (0.33)	7.00 (0.64)	29.0940 (0.28)	28.0093 (0.23)	**26.6524** (0.32)	15.05 (3.00)
16	224.3123 (5.72)	32.1965 (0.24)	30.7937 (0.33)	**30.4287** (0.29)	7.20 (0.72)	28.2433 (0.28)	27.1363 (0.19)	**25.5760** (0.19)	15.63 (2.89)
18	210.9872 (7.63)	31.5826 (0.24)	30.5182 (0.29)	**30.1348** (0.27)	7.32 (0.81)	27.6008 (0.27)	26.5601 (0.20)	**24.8811** (0.16)	16.78 (3.61)
20	197.1772 (7.99)	31.2682 (0.22)	30.3116 (0.31)	**30.0384** (0.28)	7.57 (0.76)	27.1091 (0.26)	26.1079 (0.23)	**24.3698** (0.15)	18.54 (3.89)
22	183.0157 (8.03)	31.0864 (0.22)	30.2344 (0.30)	**30.0739** (0.28)	8.56 (0.98)	26.6984 (0.28)	25.8048 (0.21)	**24.0129** (0.17)	21.39 (5.19)
24	172.8251 (10.59)	30.9921 (0.23)	**30.0202** (0.23)	30.1603 (0.27)	8.28 (1.41)	26.3648 (0.27)	25.4523 (0.24)	**23.7723** (0.20)	21.36 (6.42)
5	241.3032 (5.09)	117.3238 (2.22)	62.2867 (0.76)	**62.0646** (0.67)	24.59 (2.02)	58.9883 (0.53)	**58.7930** (0.56)	59.5964 (0.49)	30.82 (3.28)
7	222.1441 (4.50)	67.7577 (1.31)	46.7291 (3.92)	**46.4112** (3.73)	27.94 (1.79)	39.4703 (0.34)	**39.3817** (0.46)	39.9948 (0.25)	38.79 (4.03)
9	204.6141 (6.00)	47.3168 (0.85)	37.0224 (1.25)	**36.8904** (1.27)	18.27 (1.68)	33.9677 (0.30)	**33.2142** (0.25)	33.5907 (0.23)	32.51 (4.88)
11	189.7513 (4.62)	38.4754 (0.50)	33.4140 (0.70)	**33.1749** (0.66)	13.97 (0.71)	31.3661 (0.29)	30.3683 (0.20)	**30.2701** (0.19)	29.47 (4.70)
13	175.7382 (4.23)	34.5154 (0.32)	32.1094 (0.43)	**31.8041** (0.41)	12.79 (1.17)	29.7644 (0.28)	28.7554 (0.21)	**28.1224** (0.20)	29.94 (6.28)
15	163.1926 (4.31)	32.7069 (0.25)	31.2654 (0.35)	**30.8941** (0.32)	11.03 (1.27)	28.6966 (0.26)	27.6899 (0.20)	**26.3893** (0.25)	28.54 (6.29)
17	149.9852 (5.14)	31.8467 (0.23)	30.7699 (0.33)	**30.3664** (0.30)	8.93 (0.94)	27.9309 (0.27)	26.9097 (0.19)	**25.3794** (0.23)	28.47 (6.19)
19	139.9730 (4.32)	31.4048 (0.21)	30.5350 (0.29)	**30.0837** (0.27)	7.91 (1.08)	27.3691 (0.26)	26.3784 (0.20)	**24.7705** (0.18)	29.67 (7.37)
21	128.1830 (4.90)	31.1697 (0.23)	30.3017 (0.30)	**30.0384** (0.27)	7.60 (0.71)	26.9015 (0.26)	25.9415 (0.20)	**24.3128** (0.18)	30.05 (6.74)
23	119.5551 (4.46)	31.0421 (0.22)	**30.0627** (0.24)	30.1166 (0.31)	6.96 (0.81)	26.5346 (0.27)	25.6021 (0.21)	**23.9719** (0.21)	28.55 (7.05)
25	110.6725 (4.39)	30.9772 (0.23)	**29.9450** (0.21)	30.1393 (0.24)	6.68 (0.89)	26.2126 (0.26)	25.2289 (0.21)	**23.7773** (0.25)	25.59 (6.02)

which sub-cluster to choose can be based on the same criterion as in the agglomeration process the choice which clusters to merge, i.e. a root node is assigned to the sub-cluster where the maximum distance to a node of it is minimal.

7. COMPUTATIONAL RESULTS

The experiments have been performed on an AMD Opteron 2214 dual-core machine (2.2GHz) utilizing benchmark instances already used in the corresponding literature (e.g. [15, 11]) from Beasley's OR-Library [3, 2] originally proposed for the Euclidean Steiner tree problem. These complete instances contain point coordinates in the unit square, and the Euclidean distances between each pair of points are taken as edge costs. As performance differences are more significant on larger instances, we restrict our attention here to the 15 largest instances with 1000 nodes.

Table 1 summarizes the results obtained for various heuristics. Given are the objective values averaged over all 15 instances, where for each instance 30 independent runs have been performed, together with the standard deviations in parentheses. Considered are the two previous construction heuristics CBTC and RTC as well as the clustering heuristic C where each cluster a good root node is assigned using one of the two dynamic programming approaches d^A (restricted search space) and d^B (approximating optimal cluster roots using a correction value κ). Since d^A and d^B derive no optimal trees for a given clustering local improvement is used to further enhance their solutions.

Binary search to identify a good value for x was performed within $\frac{|V|}{2}$ and $|V|$, except when $D < 6$. In this latter case the interval bounds have been set to $\frac{|V|}{20}$ and $\frac{|V|}{8}$. In GRASP a mean μ of 0 and, after preliminary tests, a variance σ^2 of 0.25 was used, and the procedure was aborted after $l_{\max} =$ 100 iterations without improvement. The time (in seconds) listed is the over all instances averaged maximum running time of Cd^A and Cd^B, which was also used as time limit for the corresponding executions of CBTC and RTC. To verify statistical significance paired Wilcoxon signed rank tests have been performed.

Clearly, CBTC is not suitable for this type of instances, its strength lies in problems with random edge costs. The clustering heuristic outperforms RTC for every diameter bound, where the gap in solution quality is huge when D is small and becomes less with increasing diameter bound. In general, Cd^B outperforms all other heuristics significantly with an error probability of less than $2.2 \cdot 10^{-16}$. Only when the diameter bound gets noticeably loose the first dynamic programming approach Cd^A dominates Cd^B (error probability always less than $2.13 \cdot 10^{-9}$). It can also be seen that in the even-diameter case the runtime of the clustering heuristic only increases moderately with the number of levels in the height-restricted clustering. When D is odd the search for a good center edge dominates the runtime. Thus, with increasing D the clustering heuristics even get faster since the number of potential center edges to be considered, determined in the presented preprocessing step, decreases (less direct successors of the root cluster in the HRC).

We also performed test where a strong variable neighborhood descend (VND) as proposed in [11] has been applied to the best solutions of the various construction heuristics. As expected, it flattens the differences but still the BDMSTs derived from clustering heuristic solutions are in general of higher quality. On instances with diameter bounds less than approximately 10, these trees – computed in a few seconds – can also compete with results from the leading metaheuristic, the ACO from [11], which requires computation times of one hour and more.

Figure B.10: Instance p10

Bibliography

[1] Schnipsel-Jagd am Computer. www.sueddeutsche.de/panorama/546/373357/text/, November 2003. Süddeutsche Zeitung.

[2] Sichere Vernichtung von vertraulichen Unterlagen – Verfahrensregeln. Deutsche Fassung EN 15713:2009, August 2009.

[3] Office Depot. www.officedepot.com/a/browse/shredders/N=5+10725/, April 2010.

[4] T. Altman. Solving the jigsaw puzzle problem in linear time. *Applied Artificial Intelligence*, 3(4):453–462, 1989.

[5] B. T. Ávila and R. D. Lins. A fast orientation and skew detection algorithm for monochromatic document images. In *DocEng '05: Proceedings of the 2005 ACM symposium on Document Engineering*, pages 118–126, New York, NY, USA, 2005. ACM.

[6] J. Balme. Reconstruction of shredded documents in the absence of shape information. Technical report, Yale University, USA, 2007.

[7] P. Bose, J.-D. Caron, and K. Ghoudi. Detection of text-line orientation. *In Proceedings of the 10th Canadian Conference on Computational Geometry (CCCG'98)*, 1998.

[8] H. Bunke and G. Kaufmann. Jigsaw puzzle solving using approximate string matching and best-first search. In D. Chetverikov and W. G. Kropatsch, editors, *Computer Analysis of Images and Patterns*, volume 719 of *LNCS*, pages 299–308. Springer, 1993.

[9] M. G. Chung, M. M. Fleck, and D. A. Forsyth. Jigsaw puzzle solver using shape and color. In *Fourth International Conference on Signal Processing Proceedings 1998, ICSP '98*, volume 2 of *Signal Processing Proceedings*, pages 877–880, October 1998.

[10] L. Davis, editor. *Handbook of genetic algorithms.* International Thomson Publishing Services, 1st edition, 1996.

[11] P. De Smet. Reconstruction of ripped-up documents using fragment stack analysis procedures. *Forensic science international*, 176(2):124–136, April 2008.

[12] A. Faller. *Der Körper des Menschen.* Thieme, 13th edition, 1966.

[13] B. Furht, S. Smoliar, and H.J. Zhang. *Video and Image Processing in Multimedia Systems.* Kluwer Academic Publishers, 2nd edition, 1995.

[14] D. Goldberg, C. Malon, and M. Bern. A global approach to automatic solution of jigsaw puzzles. *Computational Geometry*, 28(2–3):165–174, 2004.

[15] J. Gottlieb. Permutation-based evolutionary algorithms for multidimensional knapsack problems. In *Symposium on Applied Computing. Proceedings of the 2000 ACM symposium on Applied computing*, volume 1, pages 408–414, 2000.

[16] P. Hansen and N. Mladenović. Variable neighborhood search. In Fred W. Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 145–184. Kluwer Academic Publisher, New York, 2003.

[17] C. Jacob. *Principia Evolvica.* dpunkt.verlag, 1st edition, 1997.

[18] E. Justino, L. S. Oliveira, and C. Freitas. Reconstructing shredded documents through feature matching. *Forensic Science International*, 160(2–3):140–147, July 2006.

[19] L. Kalles et al., editors. *Theoretical aspects of evolutionary computing.* Springer, 1st edition, 2001.

[20] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys. The traveling salesman problem: A guided tour of combinatorial optimization.

[21] S. Lu and C. L. Tan. Automatic detection of document script and orientation. In *International Conference on Document Analysis and Recognition – ICDAR 2007*, volume 1, pages 237–241, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[22] B.S. Manjunath, Philippe Salembier, and Thomas Sikora, editors. *Introduction to MPEG-7.* Wiley, 1st edition, 2002.

[23] W. Morandell. Evaluation and reconstruction of strip-shredded text documents. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, 2008.

[24] V. Nissen. *Einführung in Evolutionäre Algorithmen.* Vieweg, 1st edition, 1997.

[25] Z. Ognjanović, U. Midić, and N Mladenović. A Hybrid Genetic and Variable Neighborhood Descent for Probabilistic SAT Problem. In M. J. Blesa, C. Blum, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, pages 42–53. Springer, 2005.

[26] George Orwell. *Nineteen Eighty-Four.* Penguin Books, 2nd edition, 1989.

[27] M. Petri and C. Klitscher. *Scannen und optische Zeichenerkennung.* Addison-Wesley, 1st edition, 1993.

[28] M. Prandtstetter. *Hybrid Optimization Methods for Warehouse Logistics and the Reconstruction of Destroyed Paper Documents.* PhD thesis, Vienna University of Technology, 2009.

[29] M. Prandtstetter and G. R. Raidl. Meta-heuristics for reconstructing cross cut shredded text documents. In Günther R. Raidl et al., editors, *GECCO '09: Proceedings of the 11th annual conference on Genetic and evolutionary computation*, pages 349–356. ACM Press, 2009.

[30] R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 3:1389–1401, 1957.

[31] E. Schöneburg, F. Heinzmann, and S. Feddersen. *Genetische Algorithmen und Evolutionsstrategien.* Addison-Wesley, 1st edition, 1994.

[32] E.-G. Talbi. *Metaheuristics - From Design to Implementation.* Wiley, 1st edition, 2009.

[33] C.-K. Ting. Multi-parent extension of edge recombination. In *GECCO '07 London.* ACM Press, July 2007.

[34] A. Ukovich and G. Ramponi. Features for the reconstruction of shredded notebook paper. *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, 3:93–96, Sept. 2005.

[35] A. Ukovich, G. Ramponi, H. Doulaverakis, Y. Kompatsiaris, and M.G. Strintzis. Shredded document reconstruction using MPEG-7 standard descriptors. *Signal Processing and Information Technology, 2004. Proceedings of the Fourth IEEE International Symposium on*, pages 334–337, December 2004.

[36] 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.

[37] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 133–140, 1989.

[38] F.-H. Yao and G.-F. Shao. A shape and image merging technique to solve jigsaw puzzles. *Pattern Recognition Letters*, 24(12):1819–1835, 2003.