Reconstructing Shredded Documents

Razvan Ranca

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2013

Abstract

This project looks at the challenges involved in the automatic reconstruction of strip (vertically cut) and cross (both vertically and horizontally cut) shredded documents. The unshredding problem is of interest in the fields of forensics, investigative sciences, and archaeology.

All stages of the unshredding pipeline are analysed, starting from scanned images of shreds and ending with reconstructed documents. The current bottlenecks in this pipeline are identified and solutions are proposed.

The original contributions of this project include a probabilistic scoring function which outperforms the standard cost functions used in literature, a refinement upon a previously proposed, graph-inspired, search heuristic and a tractable up/down orientation method for strip-cut shreds.

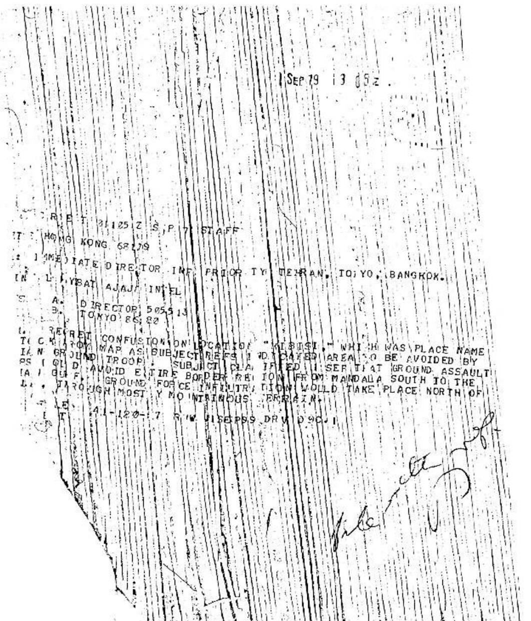
Table of Contents

Chapter 1

Introduction

Ever since the paper shredder was invented, people have worked on sticking the pieces back together again. Recently, techniques permitting the purely electronic storage and transmittal of sensitive documents have been developed but, because of convenience or for legal reasons, many sensitive documents are still printed and eventually shredded. Traditionally, the cost of reconstructing these documents was considered prohibitive as the work had to be done manually (see Figure 1.1), however with the development of methods that partly automate the process this situation is changing.

It is currently unclear what level of security the paper shredder still offers.



SEP 79 13 05z

RET 31125Z SEP 79 STAFF
HONG KONG 68279
IMMEDIATE DIRECTOR INFO PRIORITY TEHRAN, TOKYO, BANGKOK.
YBAT AJAJA INTEL
A. DIRECTOR 505513
B. TOKYO 86622
REGRET CONFUSION ON LOCATION "KISISI," WHICH WAS PLACE NAME
TOOK FROM MAP AS SUBJECT REFS INDICATED AREA TO BE AVOIDED BY
GROUND TROOPS. SUBJECT CLARIFIED I SEE THAT GROUND ASSAULT
SHOULD AVOID ENTIRE BORDER REGION FROM MANDALA SOUTH TO THE
GULF. GROUND FORCE INFILTRATION WOULD TAKE PLACE NORTH OF
THROUGH MOSTLY MOUNTAINOUS TERRAIN.
E 2 A1-120-17 R/W 31SEP99 DRV D9C.1

Figure 1.1: A shredded document belonging to the CIA. This was reconstructed by a team of carpet weavers during the 1979 Iran hostage crisis. The reconstructed documents were eventually released by the Iranian government in a series of books [23]

1.1 Importance

Techniques such as dumpster diving have long been used to gain access to sensitive information. Skoudis [44], for instance, discusses how such techniques, in conjunction with basic social engineering, can completely circumvent the security measures protecting a system. Similar approaches also account for part of the several million identity theft cases identified by the US Federal Trade Commission every year [51]. In the face of this danger, both Skoudis and the Federal Trade Commission call attention to the shredding of sensitive documents as a good security measure to take. [44, 50]. However, the development of commercial shredded document reconstruction software [48] casts doubts on the benefits of shredders and highlights the need for further research in this area.

One recent initiative looking at the reconstruction problem was the 2011 DARPA[1] shredding challenge, which offered a $50,000 prize for the first team to successfully solve a series of puzzles printed on shredded paper [11]. The puzzle was solved in 32 days, with the winning team managing to reconstruct a total of 5 documents shredded into more than 10,000 pieces (see Figure 1.3).

The DARPA Challenge was motivated by the difficulties that troops encounter in war zones, while trying to make sense of the remnants of destroyed documents. Further research in this area could have a significant impact on a number of related problems in the fields of forensics and investigative science. One of the most notable projects that could benefit from advancements in unshredding technology is the current effort to recover the shredded archives of the East German secret police. The STASI[2] destroyed most of its archives before the 1990 reunification with West Germany. These archives consist of 16,000 bags of shredded documents and, so far, it has taken three dozen people six years to reconstruct 300 of them [21]. At that rate, it would take 11,520 person-years to finish the task (see Figure 1.2).

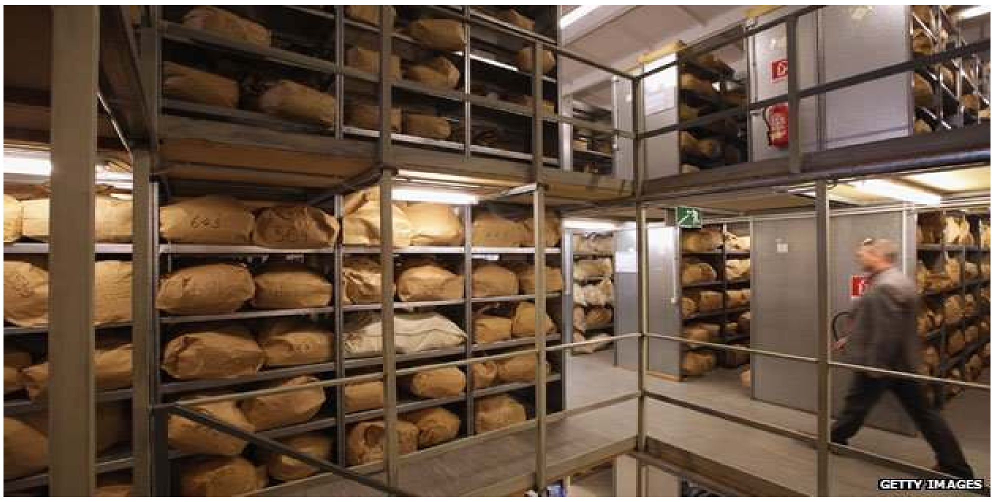
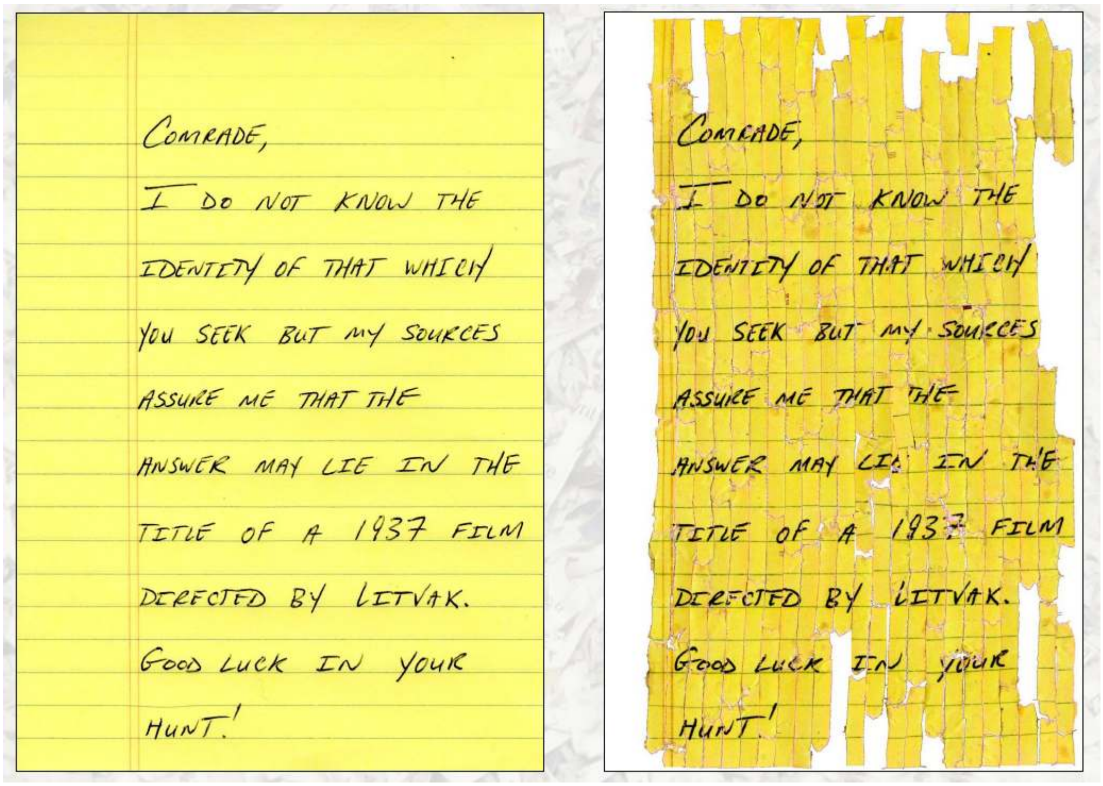


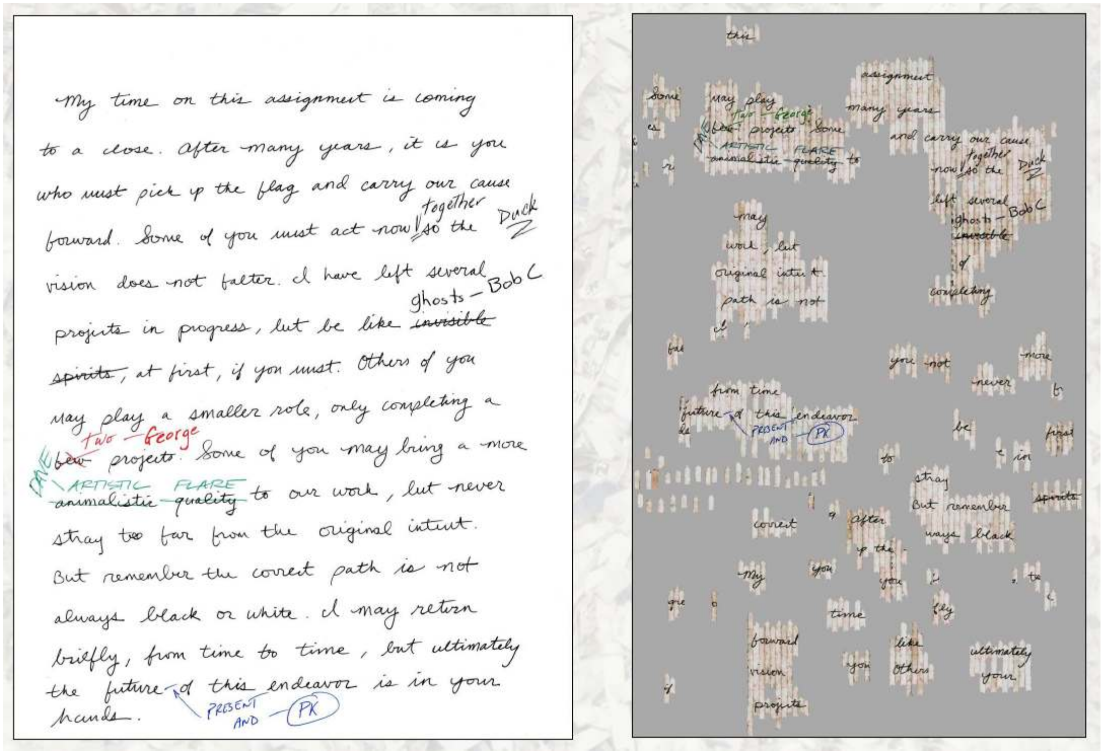
Figure 1.2: A fraction of the 16,000 bags containing the shredded STASI archives.

[1]Defense Advanced Research Projects Agency

[2]The Ministry for State Security - The official state security service of East Germany


COMRADE,
I DO NOT KNOW THE
IDENTITY OF THAT WHICH
YOU SEEK BUT MY SOURCES
ASSURE ME THAT THE
ANSWER MAY LIE IN THE
TITLE OF A 1937 FILM
DIRECTED BY LITVAK.
GOOD LUCK IN YOUR
HUNT!

(a) One of the easiest documents in the DARPA Challenge. It is perfectly reconstructed.


My time on this assignment is coming
to a close. After many years, it is you
who must pick up the flag and carry our cause
forward. Some of you must act now together so the Duck Z
vision does not falter. I have left several
projects in progress, but be like ~~invisible~~ ghosts — Bob C
~~spirits~~, at first, if you must. Others of you
may play a smaller role, only completing a
~~few~~ TWO — George projects. Some of you may bring a more
~~animalistic quality~~ ARTISTIC FLARE — DAVE to our work, but never
stray too far from the original intent.
But remember the correct path is not
always black or white. I may return
briefly, from time to time, but ultimately
the future of this endeavor is in your
hands. PRESENT AND — PK

(b) One of the hardest documents in the DARPA Challenge. Even though the reconstruction is far from complete, the competitors were able to extract the information they needed from this partial solution.

Figure 1.3: Two of the DARPA Challenge documents [11]

Outside of forensics, the techniques developed to aid the reconstruction of shredded paper could also be of use to archaeologists. Several approaches to reconstructing ancient artefacts (eg: [32, 31]) face problems regarding image feature analysis and curve matching methods which are very similar to the problems tackled in shredded document reconstruction.

1.2 Paper Shredders

Paper shredders come in many different shapes, sizes and price ranges. It is therefore worthwhile to take a closer look at the types of shredders in use and at the security they offer.
There are 3 main categories of shredders:

- **Strip-cut:** The cheapest and least secure variant. The paper is cut in long vertical strips. The width of the strips varies with the model.
- **Cross-cut:** The type of shredder typically used when extra security is deemed necessary. The paper is cut both vertically and horizontally into small rectangles. The size of the rectangles again varies with the model.
- **Other:** There are several types of industrial or specialist shredders that fall outside of the above categories. These include things such as *Grinders* which consist of a rotating shaft with blades that grinds the paper until it is small enough to fall through a mesh. These types of shredders are rarely used outside of industrial settings, so we shall not consider them further.

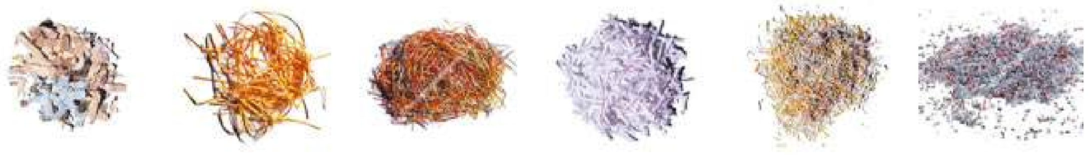
The strip and cross-cut shredders can be further classified according to the DIN[3] 32757 standard which is used throughout Europe [41]. This standard defines 6 common categories of strip or cross-cut shredders based on the level of security they offer, as shown in Table 1.1.

Security Level	Strip-Cut		Cross-Cut	
	Shred size	No. shreds/page	Shred size	No. shreds/page
Level 1	12 mm.	18	11x40 mm.	152
Level 2	6 mm.	35	8x40 mm.	216
Level 3	2 mm.	105	4x30 mm.	530
Level 4	N.A.	N.A.	2x15 mm.	2100
Level 5	N.A.	N.A.	0.8x12 mm.	6575
Level 6	N.A.	N.A.	0.8x4 mm.	19725

Table 1.1: Levels of security defined by the DIN 32757 standard

[3] Deutsche Industrial Norm

In order to get a better intuitive sense of these, Figure 1.4 shows sample shreds outputted by machines at each level.



(a) Level 1 (b) Level 2 (c) Level 3 (d) Level 4 (e) Level 5 (f) Level 6

Figure 1.4: The output generated by shredders at the different DIN levels [41].

As can be seen, the amount of security offered varies drastically. At level 1 the paper could feasibly be reassembled by hand, while at level 6, the noise introduced by the cutting and scanning of the pieces would likely make a full reconstruction impossible.

1.3 Roadmap

In Chapter 2 we formally define the problem being analysed, specify any simplifying assumptions that were made and look at the problem's complexity. In Chapter 3 we look at previously published related work. In Chapter 4 a novel probabilistic score function is proposed and shown to perform well in comparison with the standard cost functions used in literature. In Chapter 5 several tractable search heuristics are analysed and a new heuristic is proposed. Chapter 6 looks at the pre-processing function and introduces a simple and computationally efficient orientation detection method. Finally, Chapter 7 analyses the results of the whole system and discuses possible future work.

Chapter 2

Problem Definition

2.1 Domain

There are many subdomains under the general unshredding umbrella. Some of the more common distinctions look at how the paper was shredded (eg: strip-cut, cross-cut or hand torn), at the nature of the shredded document (eg: black and white or coloured; text document, picture or mixed) and at the reconstruction method (eg: fully automatic, integrated human expert or crowdsourced).

Nowadays, most documents are mechanically shredded rather than hand torn, so we decide to focus solely on the strip-cut and the cross-cut variants. Additionally, the most commonly shredded documents are black and white, typed, text documents, so this is the variant we are interested in. Finally, we only look at fully automatic solutions, partly because previous work [35, 45, 36, 40] has shown that automatic methods can be modified to incorporate user interaction with relative ease.

2.2 Assumptions

The output of a shredding device, and therefore the input to our algorithm, is a set of n shreds $S = \{s_0, ..., s_{n-1}\}$. The first part of our algorithm is a pre-processing step which aims to transform the real scanned images, containing the shreds, into images we call *ideal shreds*.
The ideal shreds have the following properties:

- Each ideal shred corresponds exactly to a separate image file. That is to say, an ideal shred image file will contain all the pixels belonging to a certain shred and only those pixels.
- All ideal shreds have the same height and width.
- All edges on all ideal shreds are either perfectly vertical or perfectly horizontal.

- All ideal shreds are correctly orientated. That is to say, if the original document were displayed with its longest edge vertically aligned, then all ideal shreds have the same orientation they would have in that original document (note that this means the document may be "upside down"; this is fine as long as all shreds share the same orientation).

- The ideal shred image files contain only pure white and pure black pixels.

The problem of reducing the real shreds to these ideal shreds is analysed in Chapter 6. For the rest of this work, the input to the algorithm is assumed to consist of ideal shreds (see Figure 2.1).

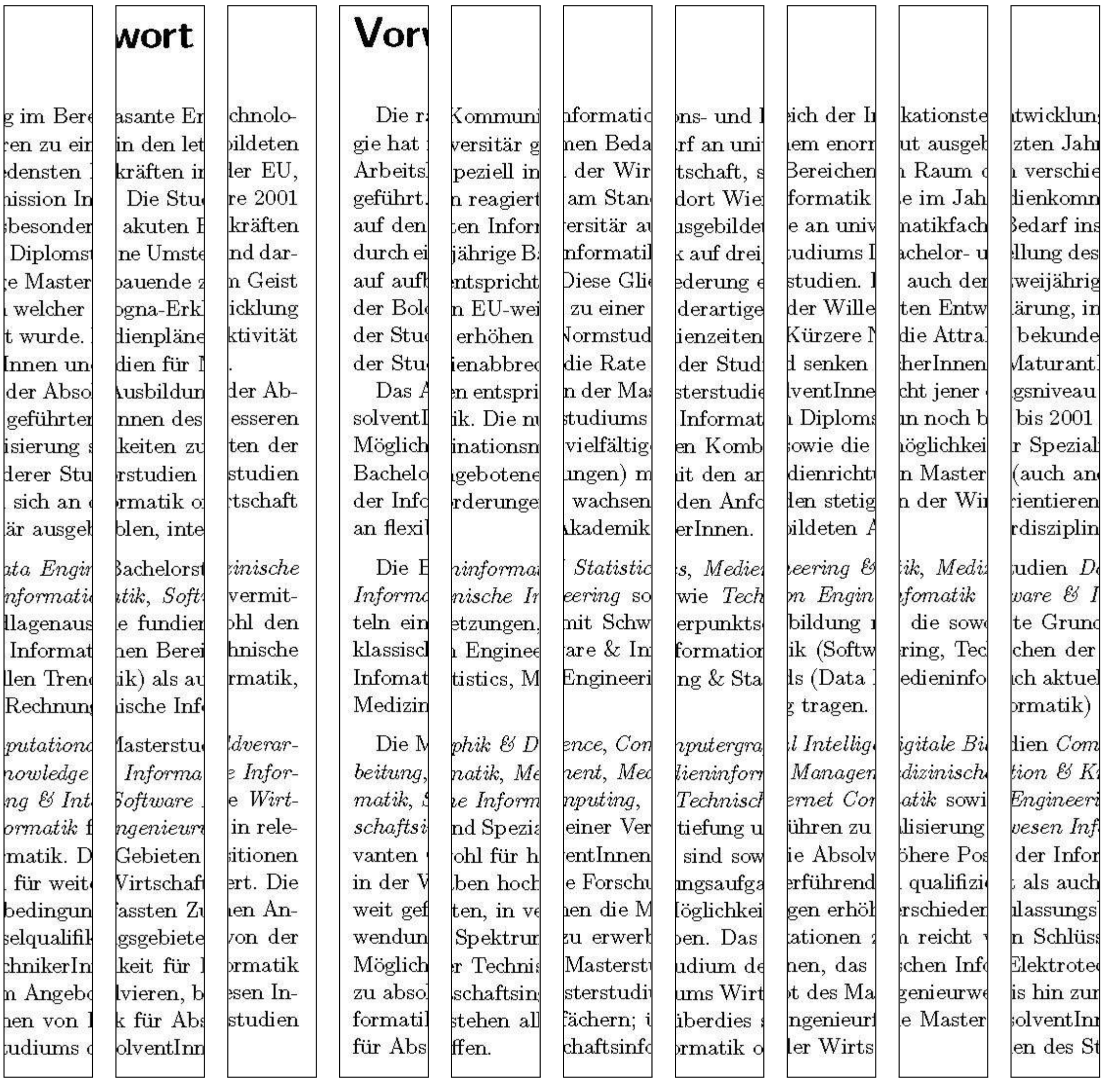


Figure 2.1: Example of *ideal shreds* of a document.

Now that we have our ideal shreds, the problem can be further subdivided into two functions.

2.3 Formal definition of score and search functions

The edge scoring function: This first function analyses all pairs of shreds for edge matches and returns a number that represents the quality of each match (see Figure 2.2).

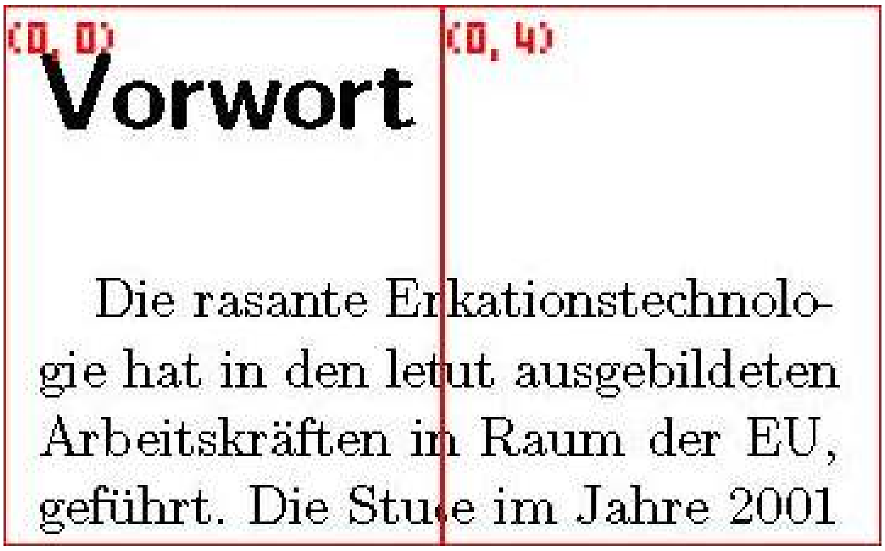

(0, 0)
(0, 4)
Vorwort
Die rasante Er kationstechnolo-
gie hat in den let ut ausgebildeten
Arbeitskräften in Raum der EU,
geführt. Die Stu e im Jahre 2001

Figure 2.2: A potential edge between two shreds. The edge scoring function has to evaluate this match and report how good it is.

Formally, we must obtain: $sc_r(s_i, s_j)$ and $sc_b(s_i, s_j), \forall s_i, s_j \in S$ where $sc_r(s_i, s_j)$ is the score of placing s_j to the right of s_i and $sc_b(s_i, s_j)$ is the score of placing s_j below s_i.

The global search function: The second function receives the scores between all pairs of shreds as input and must find an arrangement of shreds in 2D space that optimises the global score.

We consider the global score to be the sum of all individual scores between neighbouring shreds. Modifying the formulation made in [6], we can define a solution of the search function as a mapping $\Pi : D^2 \rightarrow S$ where $D = \{0, ..., |S| - 1\}$. This means each position in a two dimensional space defined by D^2 corresponds to a shred. Formally

$$\Pi(r, c) = \begin{cases} \text{the shred placed at row r, column c; IF such a shred exists} \\ \text{a completely blank shred; OTHERWISE} \end{cases} , \forall r, c \in D$$

The global score of a shred placement, $GS(\Pi)$ can then be defined as:[1]

$$GS(\Pi) = \sum_{r=1}^{|S|-1} \sum_{c=1}^{|S|-1} (sc_b(\Pi(r-1, c), \Pi(r, c)) + sc_r(\Pi(r, c-1), \Pi(r, c)))$$

[1]This assumes the shreds are indexed such as $(0,0)$ is the top, leftmost shred and $(|S-1|, |S-1|)$ is the bottom, rightmost shred

2.4 Complexity

The scoring function is forced to calculate a score for each pair of shreds. Therefore, if we have n shreds, we will have to calculate on the order of n^2 scores. Looking at the number of pieces that a page can be shred into (Table: 1.1), we can see that this quickly becomes a problem. Some techniques that could mitigate this bottleneck are explored in Section 3.4

The search function, on the other hand, has to search the space of all possible two dimensional placements of edges. Since we have no information about the original shape of the document, the space we must consider for n shreds will have $length = height = n$ and therefore $positions = n^2$. This means we are placing n shreds into n^2 slots and so the number of possible placements is $\binom{n^2}{n} = \frac{n^2!}{n!(n^2-n)!}$. Clearly this is a huge search space and, in fact, [35] has shown that the search problem is NP-hard even if restricted to just strip-cut documents. Again, looking at Table 1.1 we can easily see that optimal search solutions are not feasible for this problem and we must therefore make do with heuristics.

2.5 Modularity

By splitting the reconstruction into the three mentioned sub-functions (pre-processing, score and search), we attempt to solve the problem in a modular way.

Any of the three functions could be enhanced, or replaced entirely without needing to make any modifications in the other two functions. This is particularly useful since each function has quite a different goal and utilises different techniques (for instance, all optical character recognition related algorithms will be restricted to the score function). This separation of concerns allows us to improve algorithms more easily or extended them to different problem domains.

This modularity is something that was aimed at throughout, so that not only can any of the functions be replaced, but they can easily be composed with other functions. These factors will be discussed more thoroughly in the chapters corresponding to each function.

Chapter 3

Literature Review

3.1 Related Problems

The unshredding problem can be viewed as a special case of the general jigsaw reconstruction problem. Automatically solving jigsaws has been an active area of research since the 60s [16] and as such there is a wealth of information published on this topic. However the majority of early methods focus entirely on the shape information of the pieces (eg: [16, 18]) and are therefore not applicable to our problem. More recently there have been several attempts that also utilise the visual information of the pieces (eg: [12, 52]), however these methods mostly focus on the distribution of colours on the images and thus are not easily applicable to our black and white domain. Additionally, the solutions proposed are generally restricted to very small problem instances (usually less than 50 pieces), which is too small a number for our purposes. One promising method is presented in [30]. Here the authors reconstruct the jigsaw while looking only at image features, and also manage to solve instances with as many as 320 pieces. The features used in their comparison function are viable candidates for shred comparison or clustering.

Another related problem is the reconstruction of full-colour shredded documents. As opposed to the jigsaw puzzle formulation, this problem has no edge shape information to use. However, the availability of full-colour means that the amount of information available on a shred is much higher than in the case of binary data (i.e. black and white pixels). A thorough review of this problem is presented in [43]. Here, Skeoch analyses several types of distance functions that look at how similar two pixels are and uses these to design some simple yet effective shred comparison functions. However, the author notes that these methods don't work very well on black and white documents. Additionally, the analysis is restricted to the strip-cut variant.

Yet another related problem, is the reconstruction of hand-torn documents. In this problem, edge shape becomes once more a big factor, especially since hand-torn documents tend to contain much fewer shreds than mechanically torn ones. Most approaches (eg: [25, 7, 38]) focus on curve matching methods and are therefore not applicable to us. Additionally, since the search space is smaller, such approaches are generally quite computationally expensive ([25], for instance, was only tested on up to 15 shreds). One interesting method in this area was presented by De Smet in [13]. Here, he proposes a search algorithm that takes advantage of the way humans tend to stack hand-torn shreds and which successfully manages to speed up the search process. A method more applicable to our problem is analysed in [14], where the authors look at the effectiveness of different features when used to cluster the shreds based on similarity.

A final related problem is the *semi-automatic* reconstruction of shredded documents. Some of the most successful reconstruction methods developed so far, including those that won the DARPA Shredder Challenge [17, 9] fall under this category and incorporate human input into the evaluation loop. Using even a small amount of human expertise can vastly reduce the difficulty of the problem, because humans can catch errors early, before these have a chance to propagate. One example of such a system is shown in Figure 3.1

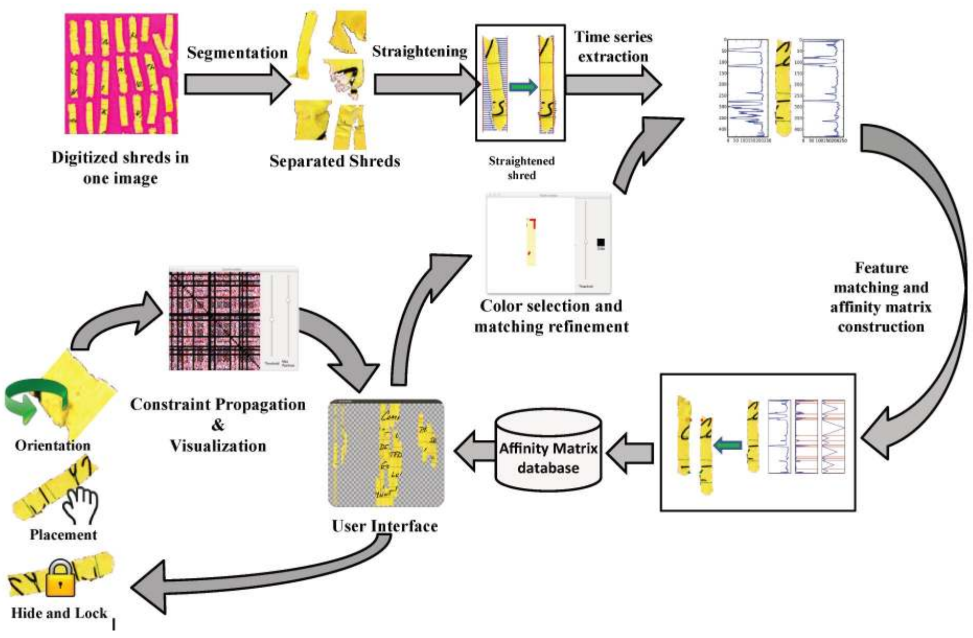

Segmentation
Straightening
Time series extraction
Digitized shreds in one image
Separated Shreds
Straightened shred
Color selection and matching refinement
Feature matching and affinity matrix construction
Constraint Propagation & Visualization
Orientation
Placement
Hide and Lock
User Interface
Affinity Matrix database

Figure 3.1: The pipeline of the Deshredder algorithm and user interface. Figure taken from [9].

A particularly interesting semi-automatic approach was developed in [53], where the authors show a shred to the user and then allow him to draw what a portion of the neighbouring shred might look like. The user is then shown shreds which match his drawing, at which point he can either select a correct match from the proposed edges or decide change his drawing.

3.2 Edge comparison functions

Most previously published approaches have used a *cost* function to compare edge matches. A cost function means that a lower cost represents a better match and, in particular, a cost of 0 shows a perfect match and a cost of ∞ shows an impossible one.

Relatively little progress has been made in developing the cost function. Most papers (eg: [35, 36, 34, 40, 6, 39]) settle on a simple "Gaussian cost" formulation which does a weighted difference of each pair of adjacent pixels on either side of the proposed join and increases the cost of the join if the pixels are too dissimilar. The logic here is that neighbouring shreds are more likely to have matching pixels on their edges. In order to mitigate the effect of noise, a "Gaussian" comparison is done by looking at a small neighbourhood of pixels rather than doing a single pixel to pixel comparison (see Figure 3.2).

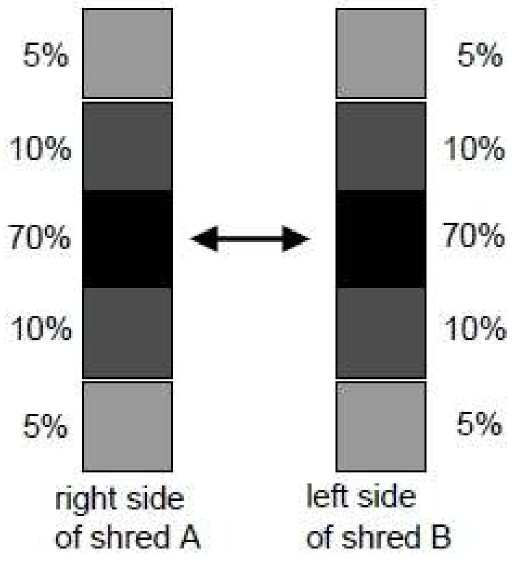


Figure 3.2: Cost function calculating the weighted sum of the difference between two opposing pixels and their respective opposing neighbours. Figure taken from [6]

In order to formally define this function, we first must get a term for the weighted sum of the yth pixel on shred A being matched to the corresponding yth pixel on shred B. We call this function $e'_h(A,B,y)$ and define it as:

$$\begin{aligned} e'_h(A,B,y) = |&0.7(v_r(A,y) - v_l(B,y)) + \\ &0.1(v_r(A,y+1) - v_l(B,y+1)) + \\ &0.1(v_r(A,y-1) - v_l(B,y-1)) + \\ &0.05(v_r(A,y+2) - v_l(B,y+2)) + \\ &0.05(v_r(A,y-2) - v_l(B,y-2))| \end{aligned}$$

where $v_r(A,y)$ returns the yth pixel on the right edge of shred A and $v_l(A,y)$ returns the yth pixel on the left edge of shred A.

Once we have this weighted sum, we can pass it through a threshold τ in order to obtain the pixel cost:

$$e_h(A,B,y) = \begin{cases} 1 \text{ if } e'_h(A,B,y) \geq \tau \\ 0 \text{ otherwise} \end{cases}$$

And finally, the edge cost is simply the sum of all pixel costs:

$$c_r(A,B) = \sum_{y \in \text{Edge Pixels}} e_h(A,B,y)$$

One problem with the above formulation is that matching a completely white edge to another completely white edge will give a perfect cost of 0. Therefore whenever such a white-on-white match is available it will be taken, which can result in undesirable behaviour (see Figure 3.3).

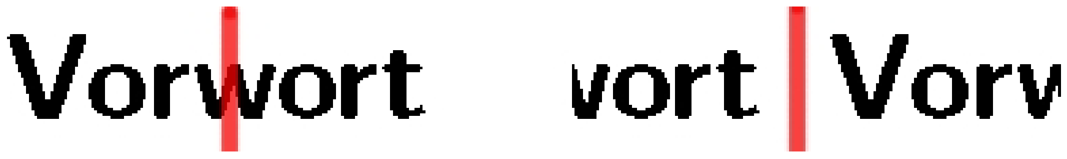


Figure 3.3: **Left:** the correct match. **Right:** white-on-white match which would get a perfect cost under the Gaussian cost function and be incorrectly preferred

A refinement to the cost function that solves the white-on-white problem is proposed in [45]. Here, the authors base the cost only on how well the black pixels match (i.e they disregard matching white pixels from their cost computation). This paper also introduces a heuristic that penalises edges that have too few black pixels on them. These improvements result in the best cost function published so far and therefore the main benchmark against which we will compare our method.

The authors of [33] present another proof-of-concept cost function improvement. They use computer vision techniques to identify individual characters that were split between different shreds and try to match them together. The authors learn the correct character shapes from the shreds themselves, therefore the method can be applied on any character set. The paper has promising preliminary results but doesn't provide comparisons with other cost functions, so aspects such as robustness to noise remain unclear.

3.3 Search functions

Significant effort has been made towards finding a good search function and several different approaches have been explored. In [35], the authors show that the search problem is NP hard by reducing it to a Travelling Salesman Problem. The authors then solve this Travelling Salesman Problem by using the Chained Lin-Kernighan heuristic [2]. In [34] an exact solution is attempted via Integer Linear Programming. Due to the large search space, this paper only looks at the strip-cut variant. The method presented here yields very good results but is intractable for any number of strips above 150. This suggests that any attempts to use Integer Linear Programming for the cross-cut variant would likely be futile.

Several attempts at using top-down local improvement have also been made. These methods first obtain a seed solution using a heuristic function and then attempt to improve upon this solution via local search methods. In [36] the authors apply the Variable Neighbourhood Search meta-heuristic [28], which contains two parts, a local search function called variable neighbourhood descent and a perturbation function which aims to help the local search escape local optima. The variable neighbourhood descent searches neighbourhoods of increasing size for a solution that is better than the current one. Here neighbourhoods are obtained by either swapping single pieces, or shifting groups of pieces around. The perturbation function simply adds some randomness in the form of occasionally switching the solution to a neighbour even if the neighbour's score is worse than our current score. In the same paper, the authors also define an Ant Colony Optimization solution [15]. This Ant Colony Optimization formulation proved to be overall better than the Variable Neighbourhood Search heuristic, but at the cost of a longer runtime. These results are improved upon in [40], where the authors use the same Variable Neighbourhood Search formulation, but embed it into a genetic algorithm. The method defines several mutation and recombination operators which are applied to an initial population of solutions generated by some simple heuristics. Between every step of the genetic algorithm, the solution pool is improved by running the Variable Neighbourhood Search algorithm.

As mentioned above, [36, 40] both rely on heuristic search methods to provide them with initial seed solutions. One of these methods is the Row Building Heuristic (RBH). RBH tries to take advantage of the fact that shreds with a completely white edge probably belong on the margin of the document. RBH therefore randomly selects a shred with a white left edge as its starting shred. It then adds the best match, as defined by the cost function, to the right of this starting edge. The greedy process is repeated, thus increasing the length of the current row, until we append a piece which has a white right edge, at which point the algorithm simply restarts and attempts to build the next row.

The second heuristic worth mentioning is the Prim Based Heuristic, which is analogous to the Prim minimum spanning tree algorithm [37]. This works by picking a random shred as a starting point and then expanding it by adding a shred to one of the four possible neighbouring positions. The process is repeated, always greedily adding the best edge to one of the neighbours of our current solution (see Figure 3.4).

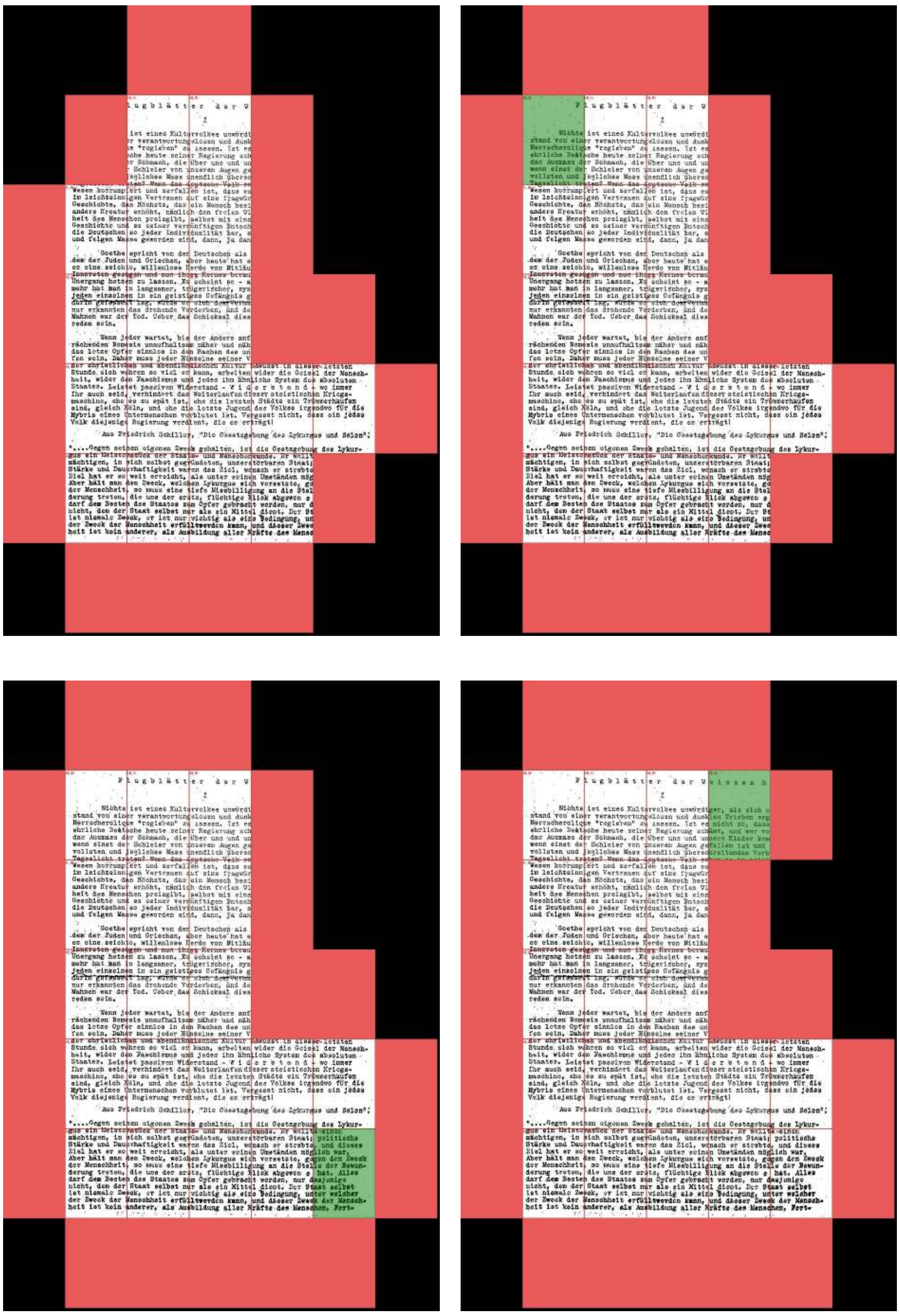


Figure 3.4: Four steps from the middle of a Prim search are shown. The green pieces highlight the changes that occur in every step and the red pieces show the positions that are considered for the insertion of new shreds.

Lastly, a heuristic called *ReconstructShreds* is proposed in [45]. This heuristic can be viewed as a relaxation of the Prim heuristic such that we can have multiple clusters of pieces at any time. ReconstructShreds is looked at in more detail in Section 5.2.

3.4 Pre-processing the shreds

As discussed in Section 2.2, a pre-processing step is needed in order to transform the real, noisy, input into a form suitable for the rest of the algorithm. Several aspects of this pre-processing step have been previously analysed in literature, namely: segmentation, skew correction, up/down orientation and clustering.

Segmentation: The first issue is to actually extract the shreds from the scanned image and to make sure they're all the same size. Skeoch [43] tackles this problem by first thresholding the original image as to detect the position of the pixels sufficiently different from the background colour. Several ways of automating the threshold selection method are analysed but ultimately, due to the high variance present in scanned images, user input is still required. After thresholding, Skeoch fits rectangles to all pixel blobs over a certain size and thus obtains the individual shreds. A complication can arise when processing narrow strip-cut documents as the shreds can have a tendency to curve, in a manner similar to human hair. Since this curvature can trip up the rectangle fitting, Skeoch proposes instead to detect the corners of the shreds and then fit a polynomial to the edges. An alternative that may be preferable if noise is a major issue, would be to use the Generalised Hough Transform [5] to detect the delimitation of each shred.

Skew Correction: If we have already detected the shred's bounding rectangles, then fixing the skew of the image can be done by rotating the rectangles so that their longest side is vertical. Alternatively, in [9] the authors take a different approach by fixing the skew first. They frame the skew correction as an optimization problem and seek the rotation that would minimize the distance between each shred pixel and a vertical line. This completely avoids the need to fit or detect the lines of the shreds since, after the shred has been oriented vertically, a simple bounding box can be taken around it. Both approaches work reasonably well in practice, with maximum errors in the range of 1-2 degrees.

If accuracies greater than this are required, then more complex document orientation methods can be employed. For instance, in [4], the authors use a row detection method and then predict the skew and orientation based on some features of these rows. Their method has a precision of 0.1 degrees, and similar results have also been obtained by several other algorithms (eg: [1, 49]). However none of these methods have been tested on extremely small shreds. Their performance degradation as the information on a shred decreases is unclear and may pose a problem.

Up/Down Orientation: If we have opted for the simpler methods of performing skew correction, then the shreds will now be either correctly oriented or rotated by 180 degrees. Several reliable methods have been developed to deal with this problem. If we restrict the problem to documents written in Roman script, then a robust method is presented in [10]. The author obtains 100% accuracy even on documents degraded by noise. Additionally, the method is employed for shred orientation detection in [14], thus showing the method can handle orientation on smaller pieces of text (though that paper handles hand-torn documents, so the shreds they look at aren't as small as the ones we might be interested in). For a completely general approach, there are also several orientation methods that work on both Roman and non-Roman scripts, usually by employing some learning algorithm (eg: [3])

Clustering: One last pre-processing task, useful in some domains, involves clustering the shreds. The problem is that often the shreds of many different documents will be mixed together, thus making the reconstruction process intractable. However, if not all the documents are completely uniform, then the search space can be reduced by clustering the shreds into similar document classes. Several image features have been proposed for this classification task. In [47], the authors propose the use of the MPEG-7 standard descriptors [42] and look at the effectiveness of features such as colour, shape and texture. The paper shows encouraging results for coloured pages, but black and white documents prove to be a harder problem. Their method is expanded upon in [46], where in addition to the MPEG-7 descriptors, the authors also detect higher level features, such as whether the paper is from a notebook, by using the Hough transform [22]. In [14] the authors propose several other custom features, such as detecting whether the text is typed or handwritten and detecting the type of paper used. For text documents, good results are obtained if the text or paper colour varies, or if the writing changes from typed to handwritten. Making use of subtler features, such as differences in fonts, proved to be more difficult.

Chapter 4

Probabilistic Score

This paper proposes a departure from the previous cost function definitions, looking instead at using a probabilistic model to directly estimate the likelihood of two edges matching.

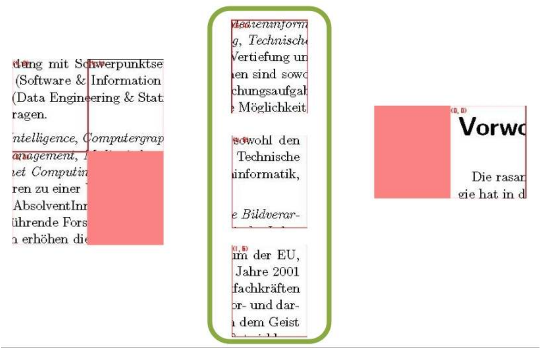
4.1 Motivation

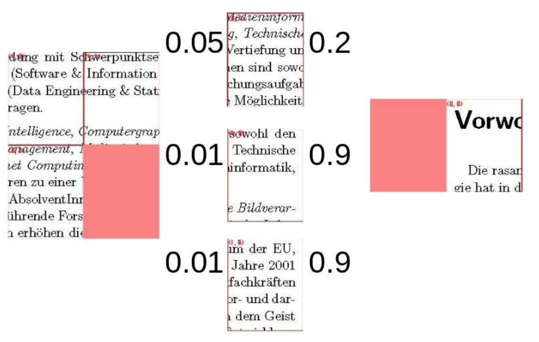
A pervasive problem with the cost functions discussed above is that their design is ad hoc and relies on hand-picked values based on the authors' empirical observations. For instance, in [35] the authors have to decide on the size of the Gaussian window they employ, on the weights to assign to the pixels that fall within that window and on a suitable value for their threshold function. All these parameters are given a static value but, since they are dependent on the source document, the authors would need to manually find different values for each class of documents. In [45] the authors not only face all of the above problems, but also have to decide on a threshold value for their row comparison and on another threshold regarding the minimum amount of black pixels that an edge must have in order for its matches to be considered relevant.

This ad-hoc formulation suffer from several impediments which the probabilistic method manages to avoid or at least ameliorate:

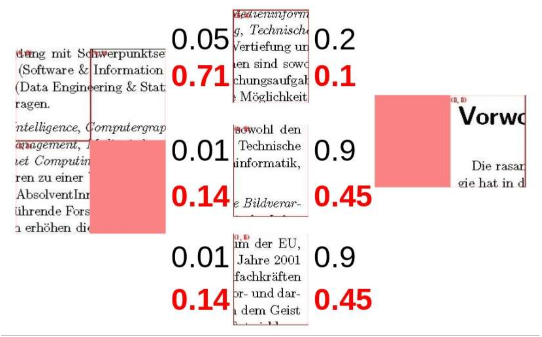
Cost function	Probabilistic score function
Relies on the values the authors hand-picked for their particular dataset.	Learns a new document's pixel distribution by analysing the shreds it is given.
Cannot be easily combined with a different similarity function.	Can be easily composed with any similarity function that produces a probability. Several such functions already exist, such as the optical character recognition based system proposed in [33]).

Cost function	Probabilistic score function
Difficult to evaluate results of the function since the numbers outputted are meaningless, only the order of the results matters.	The scores returned are estimated probabilities of a match, so the calibration of the method can be easily checked by comparing estimated and observed probabilities.
Cost is additive and must therefore be normalized relative to the sum of lengths of the matching edges.	All scores are normalized probabilities, so length of edges or number of edges matching is irrelevant (see Figure 4.1).

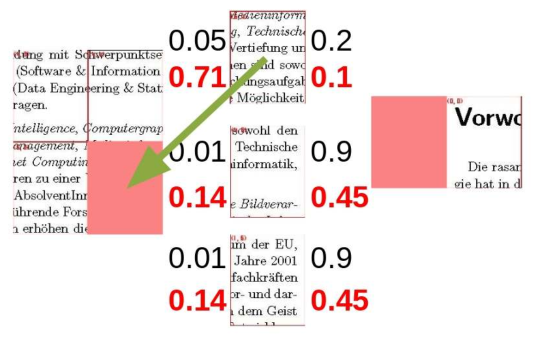

(a) Here we are trying to place one of the 3 shreds in the green rectangle into one of the two red slots. We want to identify the best of the 6 possible placements.



(b) The raw scores are shown. The scores for the left slot are lower since it has two neighbouring edges and therefore more probabilities to multiply together. For the right slot, two of the edges would give a "white on white" match and therefore a very large score of 0.9.



(c) Normalized scores shown in red. Number of pixels multiplied is irrelevant since the scores for each slot must sum up to 1. Probability mass for "white on white" matches is split between the two possible matches, thus making it less likely that any of them will be picked.



(d) The best placement is shown. Intuitively, this placement is best because its raw score is significantly larger than any of its competitor's. This score distribution means that this placement is the most likely to be correct.

Figure 4.1: Properties of normalization. Both the issue with the varying number of pixels and the "white on white" matches are solved by normalizing. In contrast, previous methods [35, 45] had to employ ad-hoc heuristics to address these problems.

Additionally, as will be shown in Section 4.3, the probabilistic score outperforms all previously formulated scoring functions.

Lastly, the effectiveness of probabilistic models, when applied to text data, has been repeatedly shown. Pixel prediction goes back to systems such as the 1981 JBIG lossless compression ISO standard (which tried to predict the value of a pixel using several features including 6 of its neighbours [20]) and is still employed in today's state of the art encoders [19].

4.2 Description

To the best of our knowledge, a probabilistic scoring function has never been previously used in this domain. We therefore choose to restrict ourselves to a relatively simple probabilistic model which can be used as a benchmark for future, more complex, approaches.

When looking at a proposed match, we try to estimate the probability that a candidate pixel is correct given several of its neighbouring pixels (called the candidate pixel's "context", see Figure 4.2). We refer to this conditional probability as $\Pr(p \mid C(p, E_x))$, where $C(p, E_x)$ is the context of pixel p when placed next to edge E_x. Ideally we'd want to learn these conditional probabilities by analysing the original document, which we obviously don't have access to. However, relatively few pixels are destroyed when shedding a document and we can assume that the ones that are destroyed are uniformly distributed over the space of contexts. Therefore the average distribution of pixels within the shreds will, in general, be a close approximation to the distribution of those in the original document. This similarity means that we can get a good estimate of the needed probabilities by obtaining the probabilities for each individual shred and then averaging them over all shreds.

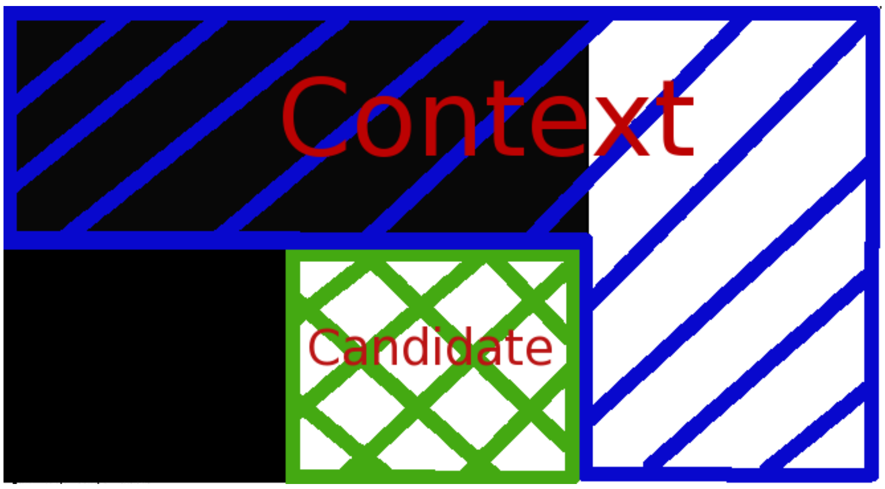


Figure 4.2: This shows a proposed match between the top 3 and the bottom 3 pixels. The model estimates the probability that the candidate pixel is white based on the four context pixels

4.2.1 Edge likelihood

Once we obtain the conditional probabilities by analysing our shreds, then the raw probability of two edges matching can be calculated by sliding the context down the proposed edge and multiplying all the individual candidate pixel probabilities (see Figure 4.3).

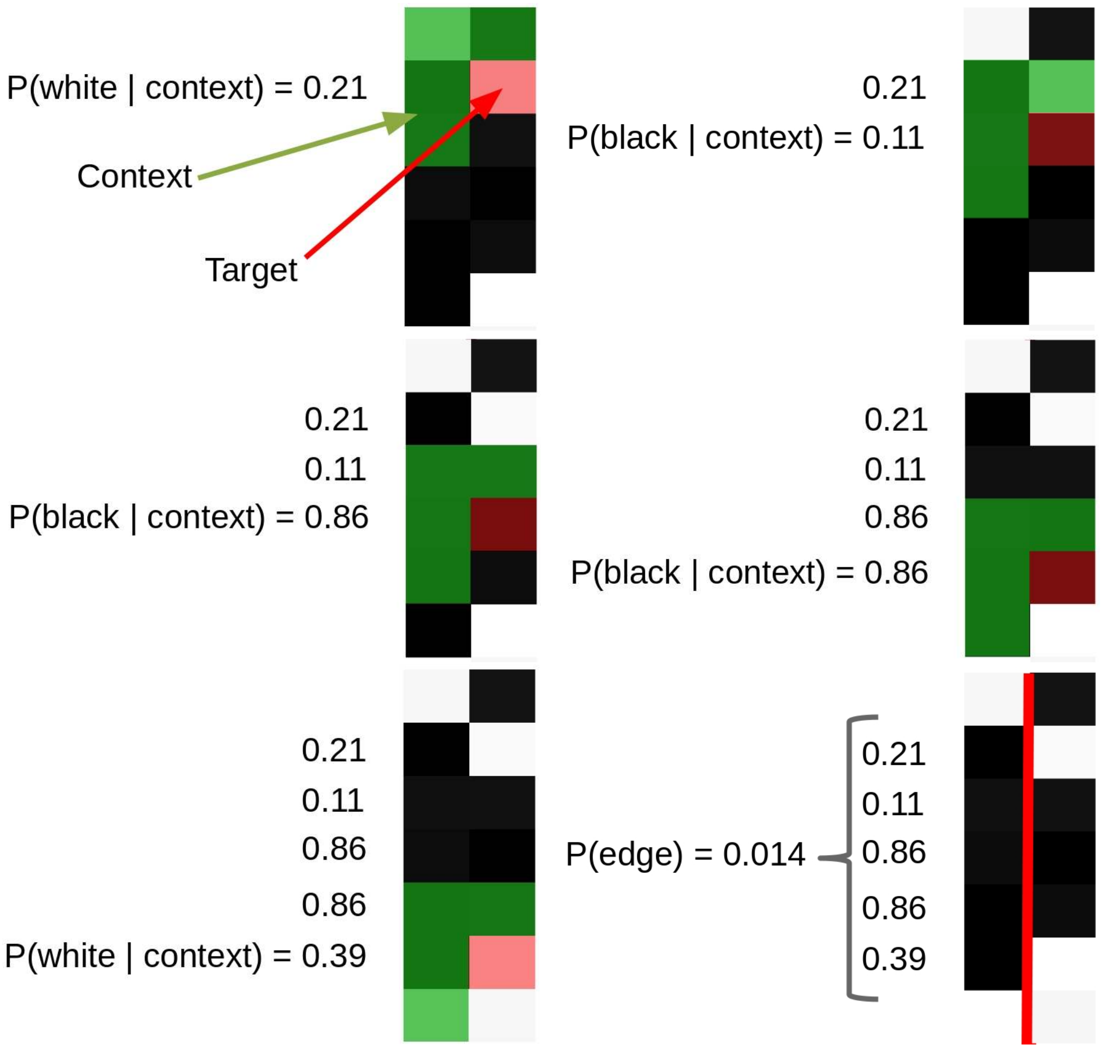

P(white | context) = 0.21
Context
Target
0.21
P(black | context) = 0.11
0.21
0.11
P(black | context) = 0.86
0.21
0.11
0.86
P(black | context) = 0.86
0.21
0.11
0.86
0.86
P(white | context) = 0.39
P(edge) = 0.014
0.21
0.11
0.86
0.86
0.39

Figure 4.3: The context slides down the proposed edge calculating a series of target pixel probabilities. All of these are then multiplied together to give the raw edge probability.

That is to say, if we have a proposed matching between edge E_1 and edge E_2, and E_1^x represents the xth pixel on edge 1, then we would estimate the joint probability of the pixels in the two edges as being:

$$\Pr(Pix(E_1), Pix(E_2)) = \prod_{i=1}^{len(E_2)} \Pr(E_2^i \mid C(E_2^i, E_1))$$

A problem arises though, because the shape of the context prohibits us from taking the conditional probability for the first and last few pixels (depending on the length of the context). For these the best we can do is assign them the prior probability, $\Pr(p)$, which is also extracted from the data simply by counting the proportion of white and black pixels. Therefore, when using the context of size 4 shown in Figure 4.2, the edge probability will actually be:

$$\Pr(Pix(E_1), Pix(E_2)) = \Pr(E_2^1)\Pr(E_2^{len(E_2)}) \prod_{i=2}^{len(E_2)-1} \Pr(E_2^i \mid C(E_2^i, E_1))$$

4.2.2 Normalization

The above, while being a good measure of the joint probability of pixels on an edge, is a function that will tend to decrease as $len(E_2)$ increases. The function will therefore tend to prefer shorter edges with less probabilities to multiply. Luckily, using probabilities gives us one more piece of information here. In the original file, every edge of every shred has only one correct match, since no two shreds are superimposed. Therefore, we know that the sum of the probabilities of all matches along one edge should sum up to one. In technical terms, if S_E is the set of all edges and $Pr(E_1, E_2)$ is the probability that edges E_1 and E_2 match, then:

$$\forall E_a \sum_{E_x \in S_E} \Pr(E_a, E_x) = 1$$

To obtain the edge probabilities that satisfy this constraint, the previous pixel probabilities are normalized along every edge. After the normalization, the preference for shorter edges is eliminated and the probabilities for different edges and pieces can be directly compared regardless of local features such as a particular edge length. Therefore, accounting for the normalization process, the final definition becomes[1]:

$$\Pr(E_1, E_2) = \frac{\Pr(E_2^1)\Pr(E_2^{len(E_2)}) \prod_{i=2}^{len(E_2)-1} \Pr(E_2^i \mid C(E_2^i, E_1))}{\sum_{E_x \in S_E} \Pr(E_x^1)\Pr(E_x^{len(E_x)}) \prod_{i=2}^{len(E_x)-1} \Pr(E_x^i \mid C(E_x^i, E_1))}$$

The normalization discussed above has an additional property, which ends up mitigating the predisposition towards whitespace that some of the other cost functions suffer from. The property in question is that if E_1, E_2 and E_3 are very similar or identical edges then they will have less chance of being picked as the best match because the available probability mass of any match will be split equally among the 3 of them. Formally, if I_{E_x} is the set of edges identical to E_x then:

$$\forall E_a \Pr(E_a, E_x) \leq \frac{1}{\mid I_{E_x} \mid}$$

[1]There actually is a further problem with the presented formula, namely that it doesn't account for edges belonging to the same piece. If E_1 and E_2 both belong to the same shred then obviously the probability that they match is 0 since a shred can't be in 2 places at once. This extra complication is ignored in the mathematical formalism for reasons of clarity

Since there are usually several white edges floating around in the edge set at any time, the above property ends up naturally discounting them without the need for any ad hoc heuristics (see Table 4.1 for a typical situation and Figure 4.3 for an additional illustration). To further discount the white on white matches we can introduce dummy white pieces into the shreds pool.

	Raw probabilities			Normalized probabilities		
	Edge 3	Edge 4	Edge 5	Edge 3	Edge 4	Edge 5
Edge 1	0.90	0.90	0.20	0.45	0.45	0.10
Edge 2	0.10	0.10	0.50	0.14	0.14	0.71

Table 4.1: Here Edge 1, 3 and 4 are all white and as such have a very good raw score when matched together. After normalization however, the match between edge 2 and 5 is considered a better bet than any match containing edge 1, as edge 1's probability mass is distributed equally among the identical edges 3 and 4.

4.2.3 Learning

Given the above descriptions, the problem of calculating the probability of any two edges matching can be reduced to the problem of estimating $\Pr(p \mid C(p, E_x))$ for an arbitrary pixel and its context. Depending on the size of the context, various machine learning methods could be used to accomplish this task. However, if the context is of the form shown in Figure 4.2, then a simple exhaustive exploration of the context space becomes a possible solution. If the context is of size 4, as above, and we are working on black and white documents, then there are only $2^4 = 16$ possible contexts. Since the number of contexts we can sample is directly proportional to the pixel count of the source image, for an average one page document, the number of observations will exceed 1 million. In practice this means that all 16 possible contexts are well represented in the document (a document where any of the contexts had less than 250 observations hasn't yet been encountered).

Of course the fit to the data will be limited by such a small context, however care must be taken when extending the context as it can easily lead to over-fitting. For instance using a context of size 7 instead of 4 usually leads to having several contexts with a number of observations in the single digits. Such values can, of course, not be trusted. Therefore it seems necessary to use more sophisticated methods in conjunction with a larger context.

One such attempt was made by using neural networks. Different architectures were tried, with both contexts of size 4 and 7, however no significant difference was found when compared with the above direct estimation. It's likely that a context of 7 was still too small for the neural networks to manage to gain an advantage over direct estimation. Therefore the neural networks were abandoned as the longer training time could not be justified on the small contexts used in this method. If, however, we were to extend the score to use a significantly larger context or an altogether different and more complex probabilistic model, then more involved learning methods would likely prove essential.

4.3 Evaluation

The evaluation measures used here work by comparing the edges predicted by the algorithm to the real edges. In order to obtain the predicted edges, for every edge, we simply take the most likely pair edge. If $PredMatch(E_a)$ is a function that returns E_a's predicted match, then:

$$PredMatch(E_a) = \arg\max_{E_x} \Pr(E_a, E_x)$$

4.3.1 Comparison with Gaussian cost functions

In order to perform this comparison, a function $CorrMatch(E_a)$ is defined, which returns the correct match for edge E_a. With this function we can define the $score(E_a)$ function as:

$$score(E_a) = \begin{cases} 1 & \text{if } CorrMatch(E_a) = \arg\max_{E_x} \Pr(E_a, E_x) \\ 0 & \text{otherwise} \end{cases}$$

However, there's a mistake here. The problem is that $\arg\max_{E_x} \Pr(E_a, E_x)$ is not guaranteed to return a single element. What we really need to check for then is $CorrMatch(E_a) \in \arg\max_{E_x} \Pr(E_a, E_x)$.

This change allows for multiple maximum likelihood matches, but doesn't account for the increased probability that the correct edge is within the predicted set by chance. In the extreme case, if the probability score returned the same value for every edge, then this scoring function would judge any pairing as correct. The solution is to discount the score based on the size of the predicted set (which also makes intuitive sense, since when a search is actually performed, if there are multiple matches with the same probability, the search will simply have to pick one at random). The final score function is:

$$score(E_a) = \begin{cases} \frac{1}{|\arg\max_{E_x} \Pr(E_a, E_x)|} & \text{if } CorrMatch(E_a) \in \arg\max_{E_x} \Pr(E_a, E_x) \\ 0 & \text{otherwise} \end{cases}$$

Therefore the proportion of correctly predicted edges (where S_E is again the set of all edges) is:

$$predictedCorrect = \frac{\sum_{E_x \in S_E} score(E_x)}{\mid \{E_a \mid CorrMatch(E_a) \in S_E\} \mid}$$

Here the denominator is just counting the number of edges that actually have matches. This is necessary because using $\mid S_E \mid$ instead would also count the outer edges which have no correct match.

Calculating analogous measures for the cost functions defined in [35] and [45] allows us to compare the three methods and shows the probabilistic score as having a relatively consistent advantage[2] over the other functions (see Figure 4.4).

[2]In the presented graph, the advantage only manifests on samples with more than 50 shreds. The performance on the small cases is very volatile, causing the results of the methods to vary from document to document. The performance on the larger instances is, however, consistently in favour of ProbScore.

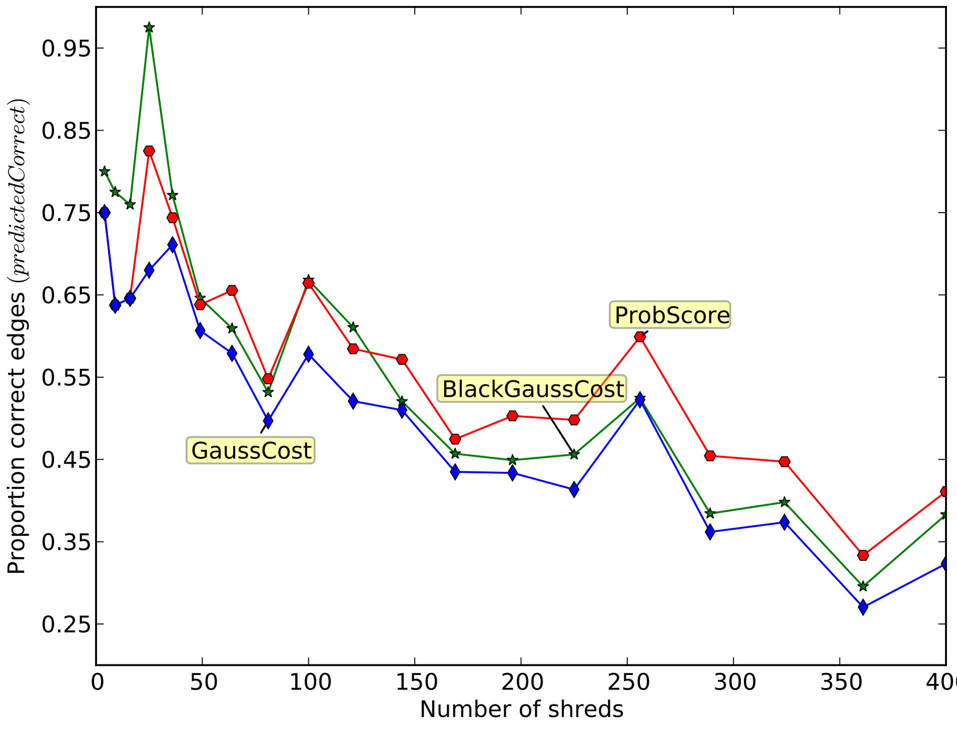


Figure 4.4: The probabilistic score has better results on medium and large instances than the cost functions presented in [35] (GaussCost) and [45] (BlackGaussCost)

4.3.2 Validity of predicted probabilities

As mentioned in the Motivation, there is another (perhaps more natural) evaluation that we can perform. Namely comparing the predicted and observed probabilities for our predicted edge matches. The observed probabilities are calculated exactly as in the previous section, and the predicted probabilities are recorded for every predicted match.

The formula $bucket = \lfloor prob * 10 + 0.5 \rfloor$ is used to place the predicted probabilities into one out of a total of 10 buckets. The observed and predicted probabilities are then averaged within each bucket, and the process is repeated for different number of shreds. The resulting graph is Figure 4.5.

4.3.3 Robustness

Another important aspect worth evaluating is the robustness of the method. When used in real life situations it is unlikely that documents will always be high-resolution, perfectly cut into shreds and completely smudge-free. In order to test this, the results obtained on an image are compared to those obtained when various types of noise are added to the image.

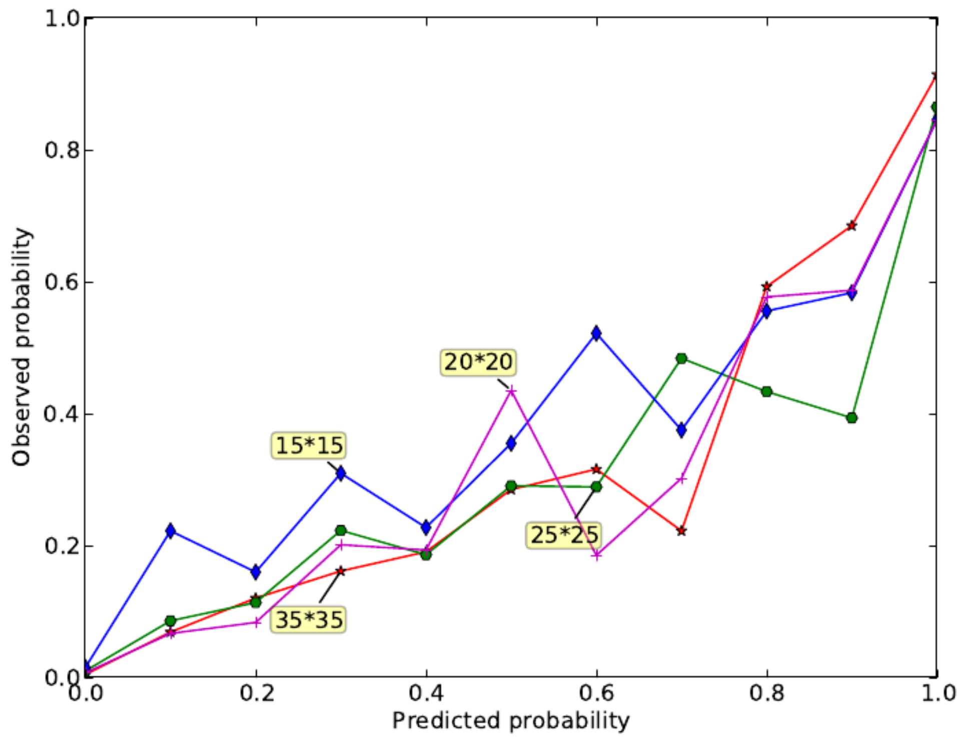
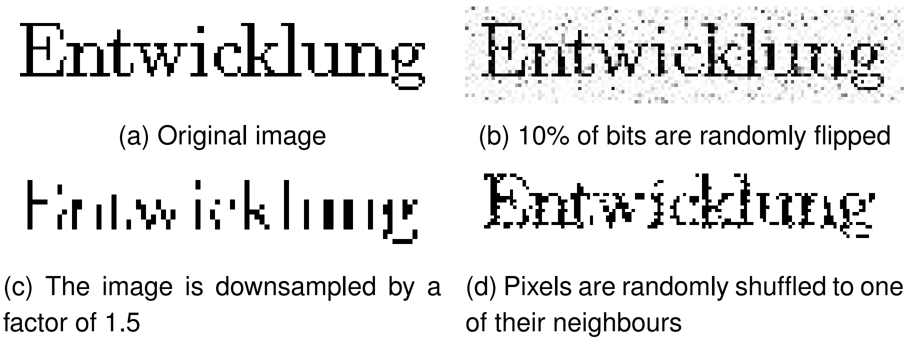


Figure 4.5: The labels show the number of cross-cut shreds. As the number of shreds increases the relationship between the predicted and observed probability degrades. However the overall shape of the curve is still generally increasing, so a higher predicted probability will usually translate into a higher observed probability and for large predicted probabilities the method is quite accurate

The types of noise analysed are: downsampling, flipping random pixels and shuffling random pixels around their original position (see Figure 4.6).



(a) Original image

(b) 10% of bits are randomly flipped

(c) The image is downsampled by a factor of 1.5

(d) Pixels are randomly shuffled to one of their neighbours

Figure 4.6: How one word of the image is modified by the various types of noise

The score evaluation is run on the original and modified documents, and the results are shown in Figure 4.7.

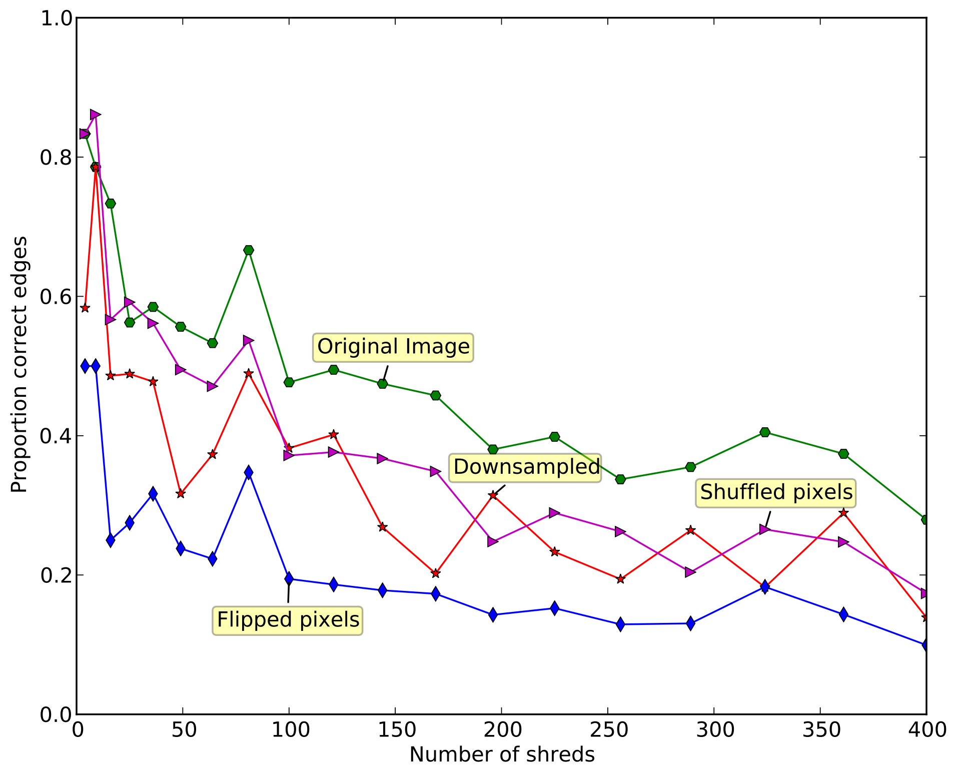
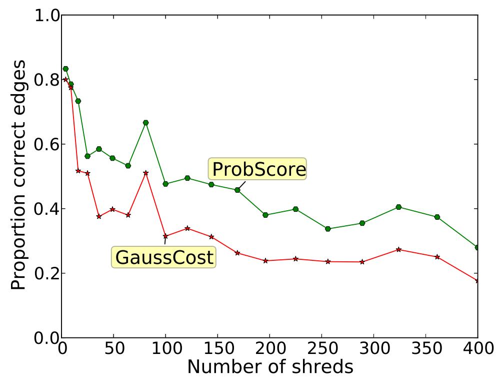


Figure 4.7: Degradation of performance between original image and 3 noisy images

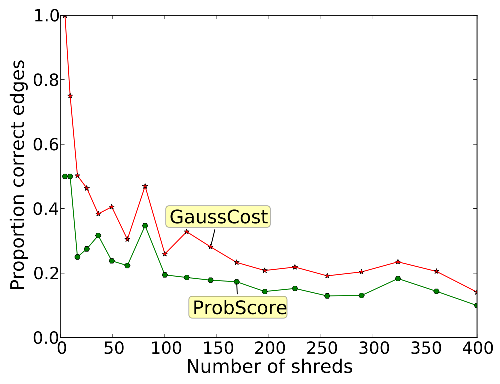
Since the probabilistic scoring function is unique for each document, as it is trained on the noisy shreds, we'd like to see if this gives it an advantage when compared to the stationary Gaussian. Therefore, the probabilistic scoring function is compared with the best Gaussian alternative on all types of noise in Figure 4.8. As expected some performance degradation is observed in all cases. It is interesting to note that the largest reduction in performance is caused by the random pixel flipping. This can be explained by the fact that the other types of noise are restricted by the existing structure of the document, if you have a section of white pixels, neither downsampling nor shuffling can introduce a black pixel into it. The algorithm seems to suffer more from the unrestrained randomness exhibited by the flipping. The good news is that in real life scenarios most noise should not be completely random and we'd expect it to be more similar to the downsampling and shuffling types of noise than to the pixel flipping.

4.4 Drawbacks

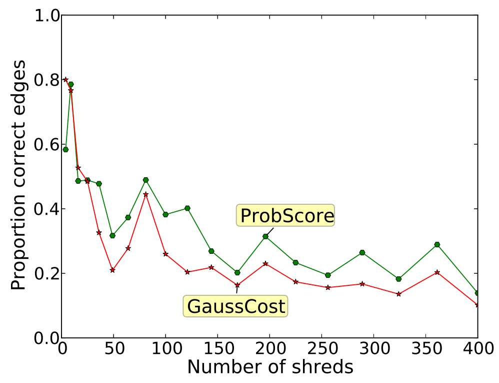
Finally, we'll look at some of the flaws in the probabilistic scoring function and at some possible solutions.



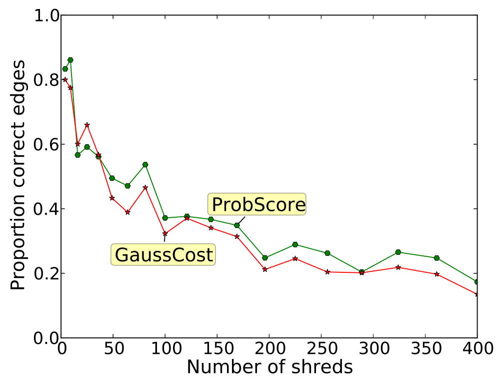
(a) Original image



(b) 10% of bits are randomly flipped



(c) The image is downsampled by a factor of 1.5



(d) Pixels are randomly shuffled to one of their neighbours

Figure 4.8: ProbScore outperforms the best Gaussian cost function on 3 out of 4 noise variants. The poor performance on case (b) is due to the simple probabilistic model not being able to cope well with completely random noise. However, this type of noise is less likely to be encountered than the other variants.

4.4.1 Uniformity assumption

Learning a single set of conditional values of the form $\Pr(p_i \mid C(p_i, E_x))$, where the number of possible contexts $C(p_i, E_x)$ is small, implicitly makes the assumption that these conditional values are relatively uniform over the whole document. This is usually a reasonable assumption to make if the documents we're interested in are all text. However this assumption is certainly not always justified. Consider the document from Figure 4.9. Here we can clearly see that there are two distinct regions, the text region and the table region. An algorithm which tries to fit a single valued model with a small context to this document cannot achieve very good results, because this document cannot be described well by such a model. The problem will only be compounded if we're looking at multiple shredded pages from possibly different documents.

One solution here is to make the context complex enough to remove the uniformity assumption, by being able to tell between the different regions in the document. This could be done by using a significantly larger context size, or perhaps by using a differ-

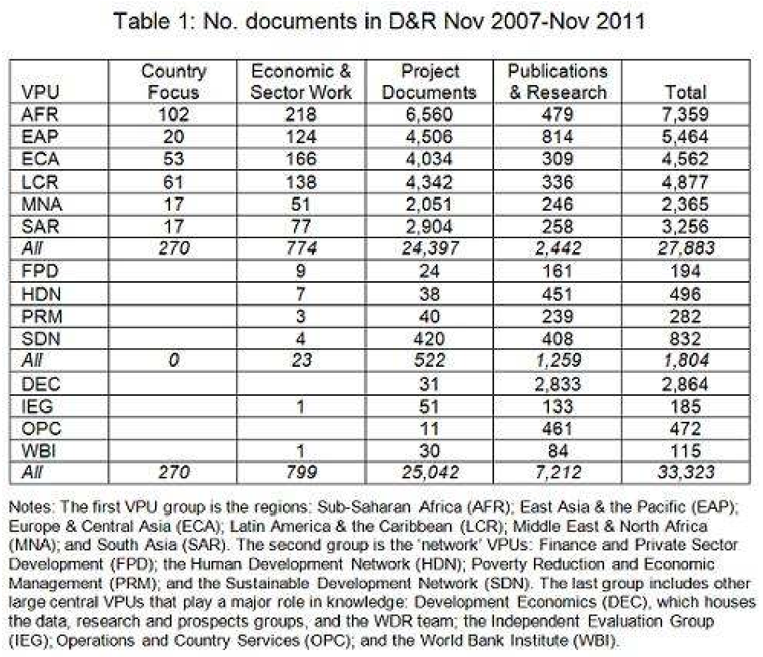


Table 1: No. documents in D&R Nov 2007-Nov 2011

VPU	Country Focus	Economic & Sector Work	Project Documents	Publications & Research	Total
AFR	102	218	6,560	479	7,359
EAP	20	124	4,506	814	5,464
ECA	53	166	4,034	309	4,562
LCR	61	138	4,342	336	4,877
MNA	17	51	2,051	246	2,365
SAR	17	77	2,904	258	3,256
All	*270*	*774*	*24,397*	*2,442*	*27,883*
FPD		9	24	161	194
HDN		7	38	451	496
PRM		3	40	239	282
SDN		4	420	408	832
All	*0*	*23*	*522*	*1,259*	*1,804*
DEC			31	2,833	2,864
IEG		1	51	133	185
OPC			11	461	472
WBI		1	30	84	115
All	*270*	*799*	*25,042*	*7,212*	*33,323*

Notes: The first VPU group is the regions: Sub-Saharan Africa (AFR); East Asia & the Pacific (EAP); Europe & Central Asia (ECA); Latin America & the Caribbean (LCR); Middle East & North Africa (MNA); and South Asia (SAR). The second group is the 'network' VPUs: Finance and Private Sector Development (FPD); the Human Development Network (HDN); Poverty Reduction and Economic Management (PRM); and the Sustainable Development Network (SDN). The last group includes other large central VPUs that play a major role in knowledge: Development Economics (DEC), which houses the data, research and prospects groups, and the WDR team; the Independent Evaluation Group (IEG); Operations and Country Services (OPC); and the World Bank Institute (WBI).

Figure 4.9: A document in which the uniformity assumption is wrong.

ent probabilistic model that takes higher level features into account (see also Section 4.4.3). A different approach would be to first split the shreds into categories and then learn a set of conditioned probabilities for every category. In theory this could also help reduce the problem size, by separating the documents into smaller uniform regions. However, avoiding overfitting might be difficult in this scenario. For instance, we wouldn't want the model to decide that every boldfaced piece of text belongs together in a separate uniform region.

4.4.2 Lack of symmetry

If E_x and E_y are two edges and E_x^{rot} and E_y^{rot} are the previous edges rotated by 180° then we might reasonably expect that

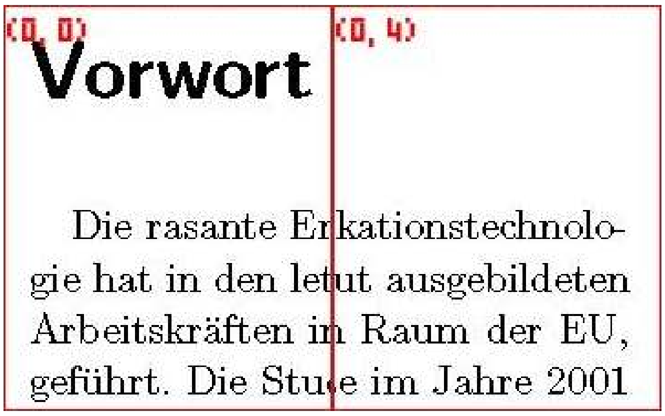
$$\forall E_x \forall E_y \Pr(E_x, E_y) = \Pr(E_y^{rot}, E_x^{rot})$$

This would be a good property to have, since we are essentially comparing the exact same edge matching in both situations. However, the current probabilistic model offers no such guarantee. In particular the probabilistic model cannot guarantee that $P(E_x, E_y) = P(E_y^{rot}, E_x^{rot})$ because it is not assured (and indeed quite unlikely) that enough data was present for these two conditional probabilities to have converged to the same value.

Depending on the situation this deficiency could be ignored. It is possible, however, that the search function being employed will assume the score to be symmetric. In that case, the best solution seems to be to calculate both $\Pr(E_x, E_y)$ and $\Pr(E_y^{rot}, E_x^{rot})$ and re-assign both probabilities to some number in between the 2 original probabilities (arguments could be made for either *min*, *max* or *avg*). Care should be taken that this re-assignment is done before the normalization step, otherwise the values will no longer add up to 1.

4.4.3 Ignoring non-edge information

This problem is common to both the probabilistic score and all the cost functions presented above. It is also the hardest to fix. Figure 4.10 shows an error made on a 5x5 cross-cut document. It is interesting to notice how even in a document cut into only 25 pieces such a convincing false match can occur. As the size of the shreds and therefore the amount of information available to our scoring function decreases further we can expect a huge number of such mistakes, making reconstruction of any non-trivial cross-cut document problematic.


(0, 0)
(0, 4)
Vorwort

Die rasante Er|kationstechnolo-
gie hat in den let|ut ausgebildeten
Arbeitskräften in| Raum der EU,
geführt. Die Stu|e im Jahre 2001

Figure 4.10: An incorrect match that would be very difficult to detect by a cost/score function which only looks at the edge pixels

The solution here likely involves combining multiple scoring functions looking at different sets of features, both higher and lower level. Luckily, as mentioned above, the probabilistic score can easily accommodate any such scoring functions as long as they can express their result as a probability. One possible higher level scoring functions is discussed in Section 4.5.

4.5 Modularity

One of the big advantages of using a probabilistic scoring function is that this allows for very easy composition with any other probabilistic function. As a proof-of-concept implementation, we propose a very simple function, called “RowScore”, and show how it can be composed with the probabilistic scoring function.

“RowScore” is based on the idea that rows in neighbouring shreds should generally match. We identify all the rows in the shreds and, when looking at a potential shred match, sum up all the discrepancies between neighbouring rows. In order to formally define this sum, we specify a function *neighRow* such that, if A is a shred and t is a y coordinate, then:

$$neighRow(A,t) = \text{the y coordinate of the row in A closest to location t}$$

If we also define $rows(A)$ to be a set of the y coordinates of all rows in shred A, the measure of how well shreds A and B match is then given by:

$$RowDist(A,B) = \sum_{t \in rows(A)} |t - neighRow(B,t)|$$

In order to translate this distance into a probability, we can make a simple Gaussian assumption. We want to assign the most probability mass to the case when the distance is 0, so this is the mean of the distribution. Therefore the final form of "RowScore" is[3]:

$$RowScore(A,B) = \frac{e^{-\frac{x^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}}$$

Using the above method, we can give the probabilistic scoring function a small but consistent boost in performance, as shown in Figure 4.11.

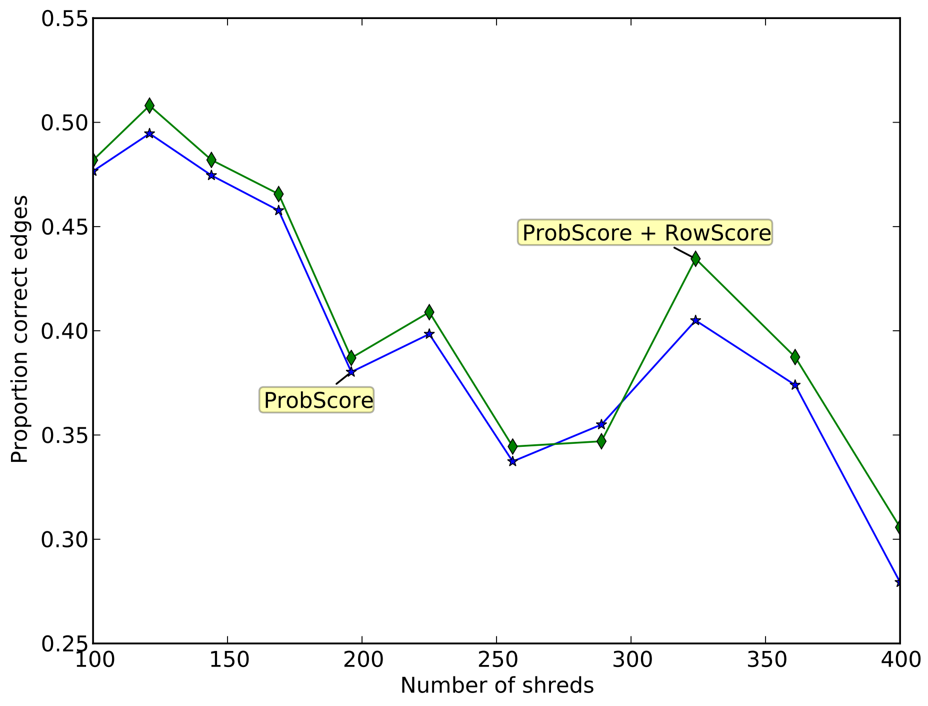


Figure 4.11: Even the extremely simple "RowScore" model gives a noticeable and consistent boost in performance to the probabilistic scoring function.

[3]The variance of the normal distribution can be set empirically, but should take account of the size of the shreds. In experiments we have seen that the performance is not very sensitive to this value.

Chapter 5

Heuristic Search

5.1 Motivation

As discussed in the Chapter 3, many complex methods have been applied to the search problem. These methods take varied approaches such as reduction to a travelling salesman problem and making use of industrial solvers [35], variable neighbourhood search [35, 36, 40], genetic algorithms [40, 29], ant colony optimization [36, 29] and reformulation as an integer linear programming problem [34].

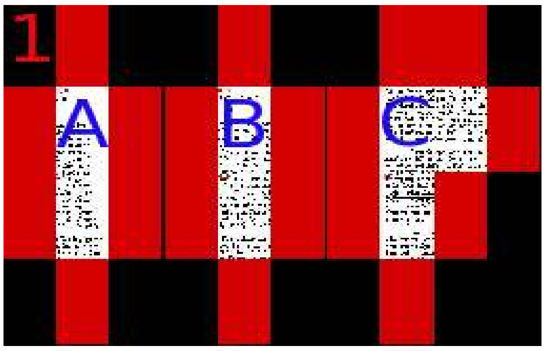
However, at least in the first instance, we decide to focus upon relatively simple greedy heuristics. Some reasons for this decision are:

- The heuristics run significantly faster than the aforementioned methods. This allows for ease of prototyping and the ability to easily experiment with many variations on both the search and cost functions.
- The inner-workings of the heuristics are transparent. This transparency allows for ease in diagnosing problems with either the heuristics themselves or with the cost functions. The transparency also allows for easy visualisation of every individual search step which enables us to observe exactly how errors occur and are how they are propagated. In contrast, using some of the previous methods would make it difficult to diagnose, for instance, whether an error is caused by the search or the cost function.
- Any advanced search methods need to be compared against a solid baseline. These heuristics provide such a baseline.
- Surprisingly good performance has been obtained using greedy heuristics. For instance, [45] reports that their simple heuristic search managed to outperform both a variable neighbourhood search and an ant colony optimization method.
- The heuristics can cope with inputs containing missing shreds or extra shreds coming from different documents. This robustness is a consequence of the heuristic's bottom up formulation, which needs to make fewer assumptions than the top down optimizing solutions.

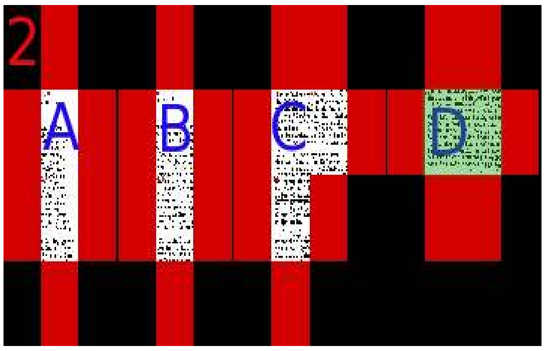
5.2 Description

Despite the fact that most of the previous work has focused on complex search algorithms, a few greedy heuristics have been explored before. We first re-implement 3 of these heuristics which, in growing order of complexity, are: the *Row building* heuristic [36], the *Prim based* heuristic [36] and the *ReconstructShreds* heuristic [45]. More details about these techniques are available in Section 3.3. Additionally, we implement a fourth, novel, heuristic which is an extension of the ReconstructShreds version.

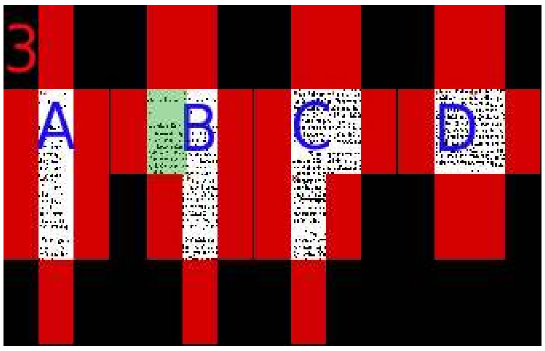
The central idea behind the ReconstructShreds heuristic is that, at each search step, the best available edge should be added to the partial solution. A few steps of the execution of this algorithm are examined in Figure 5.1, where all groups of shreds of size 2 or larger are shown (i.e. the individual pieces, which are groups of size 1, are omitted from the figure). The algorithm will continue to add the best possible edge to the partial solution and thus enlarge and merge clusters until only 1 cluster is left. This final cluster is the algorithm's final solution.



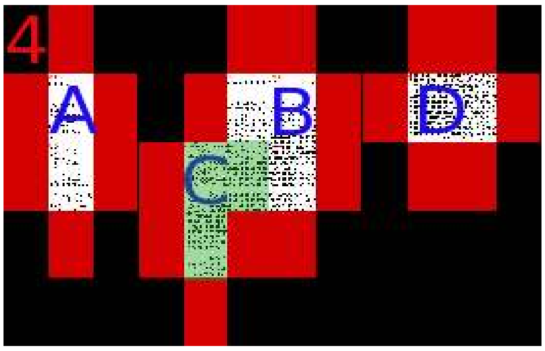
(a) There are 3 clusters: A, B and C .



(b) Best edge was between 2 shreds which belonged to neither cluster. Therefore a new cluster is created.



(c) Best edge was between a shred belonging to cluster B and one belonging to neither cluster. Therefore cluster B is enlarged.



(d) Best edge was between a shred belonging to cluster B and one belonging to cluster C. Therefore the two clusters are merged.

Figure 5.1: Four steps from the middle of a ReconstructShreds search are shown. The clusters are called A, B, C and D, the green pieces highlight the changes that occur in every step and the red pieces show the positions that are considered for the insertion of new shreds.

5.2.1 Analysis of ReconstructShreds

This algorithm calculates all the edge probabilities and goes through the list in descending order. Whenever it encounters a valid edge, it adds it to the solution set (see Algorithm 1 for the pseudocode [1]).

Algorithm 1 The ReconstructShreds heuristic

```
                                        ▷ Takes the set of edges and the set of shreds as input
 1: procedure RECONSTRUCTSHREDS($S_{edges}$, $S_{shreds}$)
 2:     $probs \leftarrow []$          ▷ Initialize 2 empty arrays for the probabilities and edges
 3:     $edges \leftarrow []$
 4:     for all $E_x \in S_{edges}$ do
 5:         for all $E_y \in S_{edges}$ do
 6:             $probs[(E_x, E_y)] \leftarrow \Pr(E_x, E_y)$ ▷ Calculate and store all the probabilities
 7:         end for
 8:     end for

 9:     $setsLeft \leftarrow |S_{shreds}|$       ▷ Initially every shred is its own set, initialize these
10:     for all $S_x \in S_{shreds}$ do
11:         $InitSet(S_x)$
12:     end for

13:     while $setsLeft > 1$ do                  ▷ Get the edges with the max probability
14:         $(E_x, E_y) \leftarrow \arg\max_{(E_x, E_y)} probs[(E_x, E_y)]$
15:         $S_x \leftarrow GetSet(E_x)$                      ▷ Retrieve the sets of these 2 edges
16:         $S_y \leftarrow GetSet(E_y)$
17:         if $S_x \neq S_y$ & $mergePossible(E_x, E_y)$ then
18:             $S_x \leftarrow Union(S_x, S_y)$             ▷ If the edge is valid, merge the two sets
19:             $S_y \leftarrow Union(S_x, S_y)$
20:             $edges.append((E_x, E_y))$
21:             $setsLeft \leftarrow setsLeft - 1$
22:         end if
23:         $probs[(E_x, E_y)] \leftarrow 0$       ▷ make sure the processed edge isn't picked again
24:     end while

25:     return $edges$          ▷ The set of returned edges describes a complete solution
26: end procedure
```

The problem with this approach is that the probabilities used are static and therefore the algorithm is completely ignorant of the formed partial solution. Its only interaction with the current state of the solution occurs via the $mergePossible(E_x, E_y)$ function which just tells the algorithm if the current solution allows for that particular merge. In particular, the algorithm takes no account of the fact that when merging 2 sets of

[1]The pseudocode in this chapter assumes the existence of a disjoint-set data structure which can perform operations such as $InitSet(x)$ (create a set with the element x in it), $GetSet(x)$ (return the set to which x belongs) and various set operations such as $Union(S_x, S_y)$ and $Intersect(S_x, S_y)$

pieces, several new edges may form. Therefore the ReconstructShreds heuristic would be happy to merge two large sets of pieces based on 1 good edge resulting from the merge even if several other terrible edges also result from that same match (see Figure 5.2).

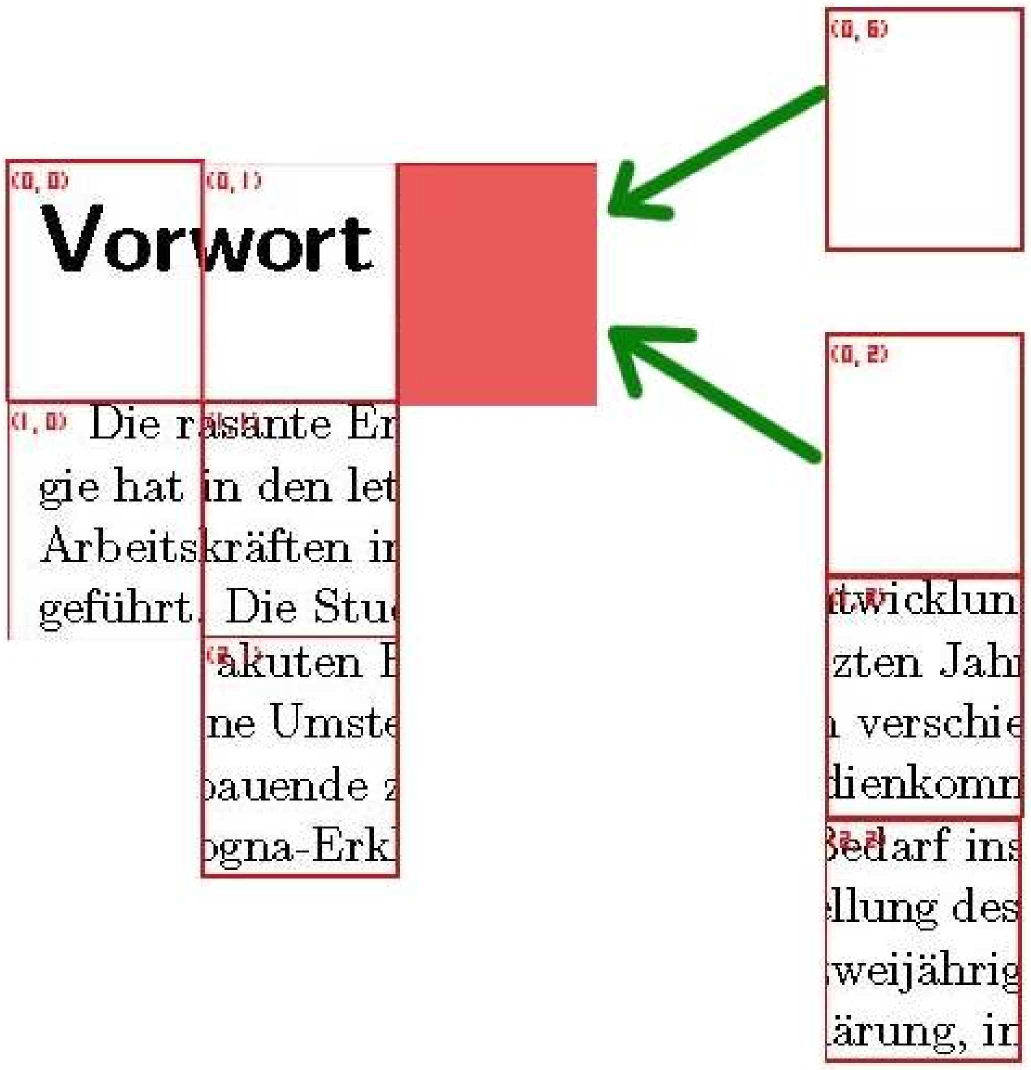


Figure 5.2: Here two potential matches for the red slot are shown. The first match is a single white piece, while the second match is a group of 3 shreds that perfectly aligns with our target shreds. Since ReconstructShreds will only look at one edge, both matches will have an identical score, namely that assigned to all white on white matches. ReconstructShreds cannot take advantage of the extra information gained since the second white shred was merged with additional shreds.

Our algorithm is designed to address this shortcoming.

5.2.2 Kruskal based heuristic[2]

Our approach towards resolving the problem observed in ReconstructShreds is to re-calculate the probabilities of two edges matching at every iteration. By doing so we can take into account all the additional matches that would result when the sets corresponding to the 2 edges are merged (see Algorithm 2).

Algorithm 2 The Kruskal based heuristic

```
1: procedure KRUSKAL($S_{edges}$, $S_{shreds}$)
2:     $edges \leftarrow []$
3:     $setsLeft \leftarrow |S_{shreds}|$
4:     for all $S_x \in S_{shreds}$ do
5:         $InitSet(S_x)$
6:     end for

7:     while $setsLeft > 1$ do
8:         $probs \leftarrow []$                ▷ Probability calculation is done for every iteration
9:         for all $E_x \in S_{edges}$ do
10:            for all $E_y \in S_{edges}$ do
11:                $probs[(E_x, E_y)] \leftarrow getProb(E_x, E_y)$        ▷ Helper function is called
12:            end for
13:            $normalize(probs, E_x, S_{edges})$    ▷ An extra normalization step is needed
14:        end for
15:        $(E_x, E_y) \leftarrow \arg\max_{(E_x, E_y)} probs[(E_x, E_y)]$
16:        $S_x \leftarrow GetSet(E_x)$
17:        $S_y \leftarrow GetSet(E_y)$
18:        if $S_x \neq S_y$ & $mergePossible(E_x, E_y)$ then
19:            $S_x \leftarrow Union(S_x, S_y)$
20:            $S_y \leftarrow Union(S_x, S_y)$
21:            $edges.append((E_x, E_y))$
22:            $setsLeft \leftarrow setsLeft - 1$
23:        end if
24:        $probs[(E_x, E_y)] \leftarrow 0$
25:    end while

26:    return $edges$
27: end procedure
```

The differences between the Kruskal and the ReconstructShreds algorithms are:

[2]This method is called a *Kruskal based heuristic* because the main goal of the method, namely that of always adding the best available edge to the solution, is analogous to the goal of the minimum spanning tree algorithm, Kruskal [26]. Therefore the general Kruskal method can be extended to this specific problem, with the only additional difficulty of having to reject potential matches which would result in 2 shreds overlapping. Indeed ReconstructShreds is already an extension of the Kruskal method, though the authors of [45] do not identify it as such

- The probability calculation has been moved inside the *while loop*, and thus our calculated probabilities need not be static any more.

- Rather than simply taking the pre-calculated probability $\Pr(E_x, E_y)$ as the final measure of the likelihood of a match between E_x and E_y, the helper function $getProb(E_x, E_y)$ is now called instead. This new function checks the proposed merge and identifies all the new edges that would be formed if this merge is selected (for convenience we assume the set of edges S_x has a function $S_x.neighbours()$ which returns all the pairs of edges that are neighbours under this merge). The final probability is then given by multiplying the individual probabilities for every newly created edge (see Algorithm 3).

Algorithm 3 The getProb helper function

```
procedure GETPROB($E_x$, $E_y$)
    $prob \leftarrow 1.0$
    $S_x \leftarrow GetSet(E_x)$
    $S_y \leftarrow GetSet(E_y)$                    ▷ Get the set of the proposed match
    $merged \leftarrow Union(S_x, S_y)$
    for all $E_a \in S_x$ do     ▷ Multiply probs of new neighbours created by the match
        for all $E_b \in S_y$ do
            if $(E_a, E_b) \in merged.neighbours()$ then
                $prob \leftarrow prob * \Pr(E_a, E_b)$
            else if $(E_b, E_a) \in merged.neighbours()$ then
                $prob \leftarrow prob * \Pr(E_b, E_a)$
            end if
        end for
    end for
    return $prob$
end procedure
```

- A normalization step is added. This is necessary because, by potentially multiplying the probabilities of several edges together to get the probability of our match, the sum of all probabilities is no longer guaranteed to be 1. Formally, after the normalization step taken in the calculation of the $\Pr(E_x, E_y)$ values (see Section 4.2.2) we had the following assurance:

$$\forall E_a \sum_{E_x \in S_E} \Pr(E_a, E_x) = 1$$

  We would like to have the same assurance regarding $getProb(E_x, E_y)$, which is why the additional normalization step is required. As before, the normalization step takes the form:

$$normProb(E_x, E_y) = \frac{getProb(E_x, E_y)}{\sum_{E_a \in S_E} getProb(E_x, E_a)}$$

5.2.3 Evaluating the heuristics

We turn to evaluating the strengths and weaknesses of the algorithms discussed above.[3] Towards this purpose we run two sets of tests.

Firstly, we look at the performance offered by the various search functions. This is accomplished by running all the heuristics on the same input document, using the same cost function and then comparing the number of correct edges observed in the output (see Figure 5.3). These results show that the Row Building Heuristic performs significantly worse than the others, but the performance of the top 3 heuristics is somewhat hard to compare.

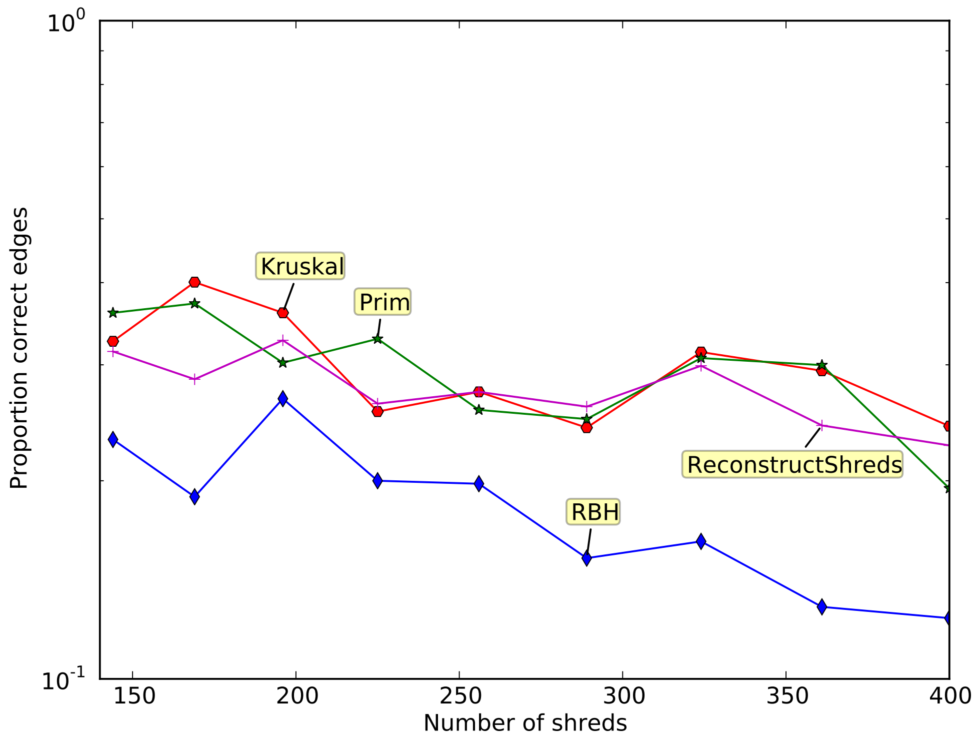


Figure 5.3: The proportion of correct edges found by each search heuristic as the number of shreds increases.

In order to try to improve upon this, a new evaluation method is proposed. This method tries to evaluate how hard it would be for a human to understand what is printed on the returned document. We therefore look at the number of moves a human would have to make to obtain the correct solution from a solution received from one of the heuristic. This assumes that the human can spot all areas of the document that are correctly reconstructed, extracts these areas and places them correctly relative to one another. Using this new evaluation method, we get Figure 5.4, which shows that Kruskal is usually the best performing heuristic, even if not by a large margin.

[3]RBH and Prim are reviewed in Section 3.3.

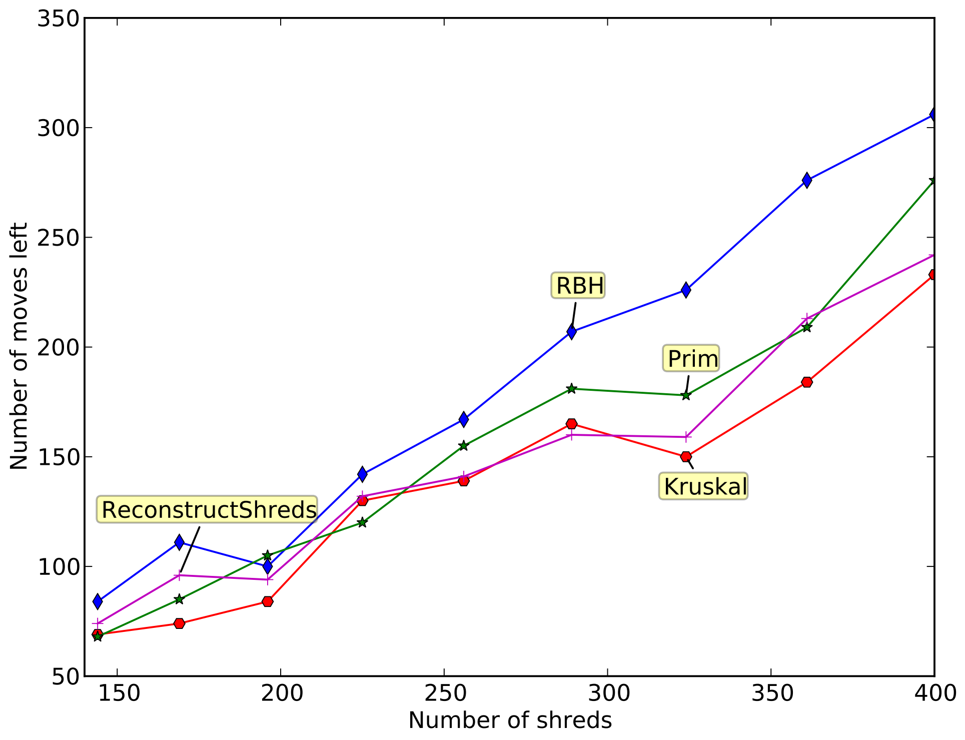


Figure 5.4: The proportion of correct edges found by each search heuristic as the number of shreds increases.

Secondly, we analyse the scalability of the algorithms. In real world scenarios an unshredder would potentially have to work with thousands, or even tens of thousands of pieces, which makes scalability an important factor to consider. For comparison purposes, the runtime of several more complex search functions are also shown in Figure 5.5.

As can be seen. even though the heuristics were not implemented with speed in mind[4], they are significantly faster than the top-down optimization methods.

5.3 Cascading

All of the greedy methods presented thus far have in common an inability to correct errors. This makes them prone to a cascading effect through which an error made early in the search process can have a major effect on the result. Since there is no means through which to move the wrongly placed shred, the search will instead try to find additional pieces which match well with said shred, and these pieces will have a significant probability of also being wrong.

[4]The heuristics are written in python, while the quoted optimization methods are written in C. Additionally, the implementation of “Kruskal” is a proof-of-concept and has not been optimized at all. A non-naive reimplementation could bring its performance more in line to that of “Prim”.

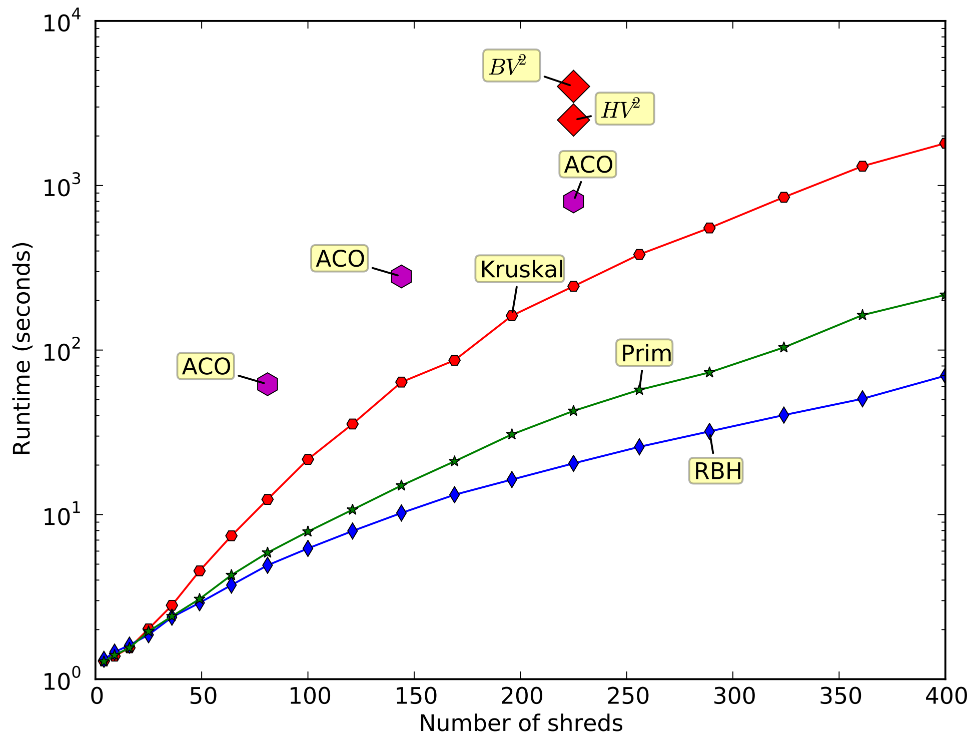


Figure 5.5: Runtime comparison for 3 of the heuristics and several previously reported methods taken from [45, 36, 40]. "ACO" is Ant Colony Optimization and "HV^2" and "BV^2" are two different genetic algorithms.

In order to quantify the magnitude of this issue it is helpful to plot the function: $Error_{search}(x)$, where $x = Error_{cost}$. This function shows the proportion of errors that the search function commits when given a score function with a certain error rate. The results of this experiment[5] (see Figure 5.6) are telling.

Even on a tiny 5 by 5 instance, in order for the Prim heuristic to achieve 80% accuracy the scoring function must be 95% accurate. As can be seen, the problem only gets worse as the number of shreds increases. On a 30 by 30 instance Prim requires the same 95% cost function accuracy rate in order to achieve a final accuracy of only 50%.

In order to address this problem, several error correcting mechanisms were analysed.

[5] This experiment was run on synthetic data. We simulated a score function with a fixed error rate and gave the output of this function to the search heuristics

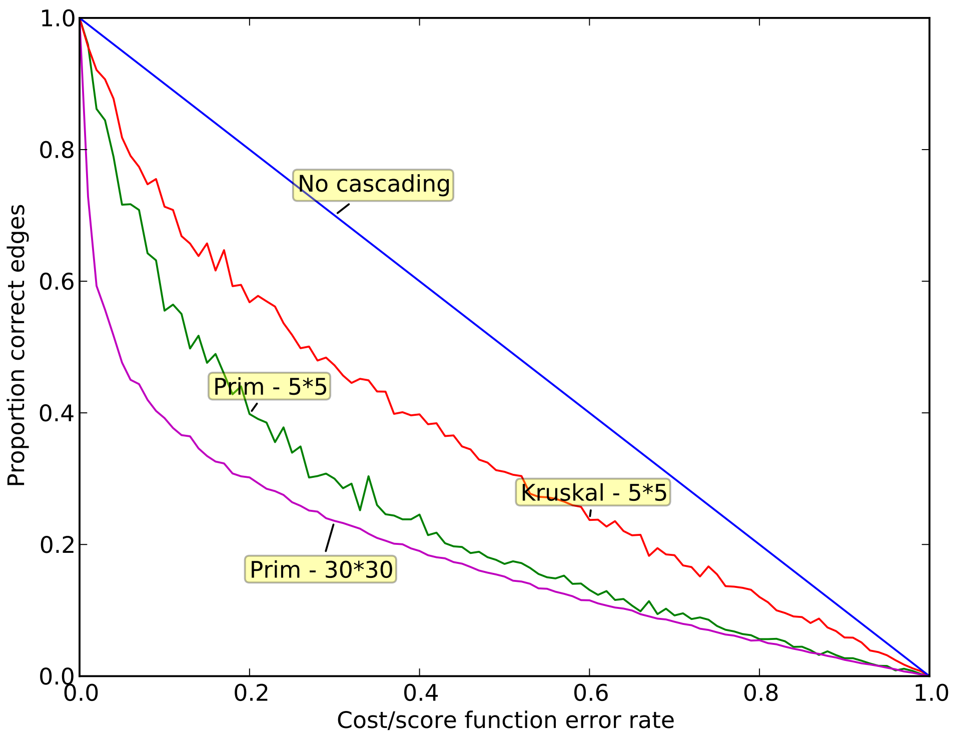


Figure 5.6: The effect the error in cost/score has on the final error of the method for both search heuristics on 5*5 shreds and for Prim on 30*30 shreds

5.3.1 Making all shreds movable

The simplest approach is to consider pieces that have already been placed as movable and treat them the same as the pieces which haven't been placed. Estimating the results of this formulation with the above cascading experiment gives us Figure 5.7.

In theory, this approach should ameliorate the cascading problem, as we are now able to fix wrongly placed pieces. However, in practice, this basic approach can lead to an infinite cycle of moves if it happens that one piece is the best match for two different edges. In that case, the greedy algorithm will continue to switch the piece between the two different edges. This specific problem can be solved by giving the greedy correction algorithm a lookahead of size one, which ensures that a piece is only moved if the piece's score would increase. This solution however will only eliminate cycles of length 2. In order to eliminate all cycles in this way, we would require a complete examination of the search tree from the current point onwards, which quickly becomes intractable.

A different solution to the cycle problem would be to use a global score rather than a local one to determine the greedy moves. Instead of looking at the individual score of matching a shred, we could check how moving that shred affects the global score of the solution. This method indeed eliminates cycles, but introduces further problems regarding the handling of outer whitespace. If we ignore outer whitespace when

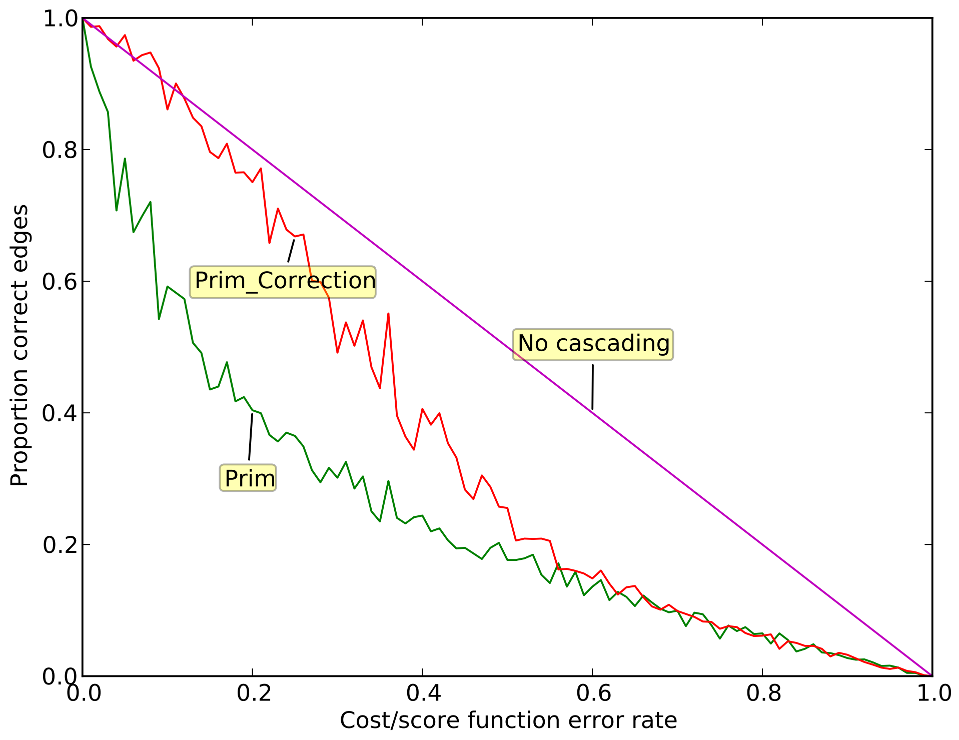


Figure 5.7: Allowing already placed pieces to move ameliorates the cascading problem. Results are on 5x5 instances.

calculating the global score, then we introduce an incentive for the shreds to be very dispersed as to minimize the amount of edges that count towards the score. Conversely, if we enforce outer white-space costs, we introduce a large amount of noise since most of the shreds analysed at the beginning of a solution won't end up having an external edge by the end.

We decide to stick to a local score and, in order to eliminate cycles while using a fixed size lookahead, we remember all previous states that the search has passed through and use these to detect cycles. Once a cycle is detected, we revert to the best state encountered within the cycle and then pick a move that breaks the cycle.

5.3.2 Adding a post-processing step

As mentioned above, a problem that plagues the heuristic search methods presented here is that while the search is in progress they cannot known which pieces will end up on an outer edge and which will be in the centre. This means we cannot take account of the score of an edge piece adjacent to white space, which in turn places no incentive on the search to find compact shapes that have less pieces on an outer edge.

We can ameliorate this problem by doing a post-processing step to the search. When all the pieces have been placed we finally know what the proposed outside edges are, so we can now apply a global score which keeps count of the external whitespace. Since the search we perform is still greedy, in order for the method to find correct solutions, it will require a large lookahead (see Figure 5.8).

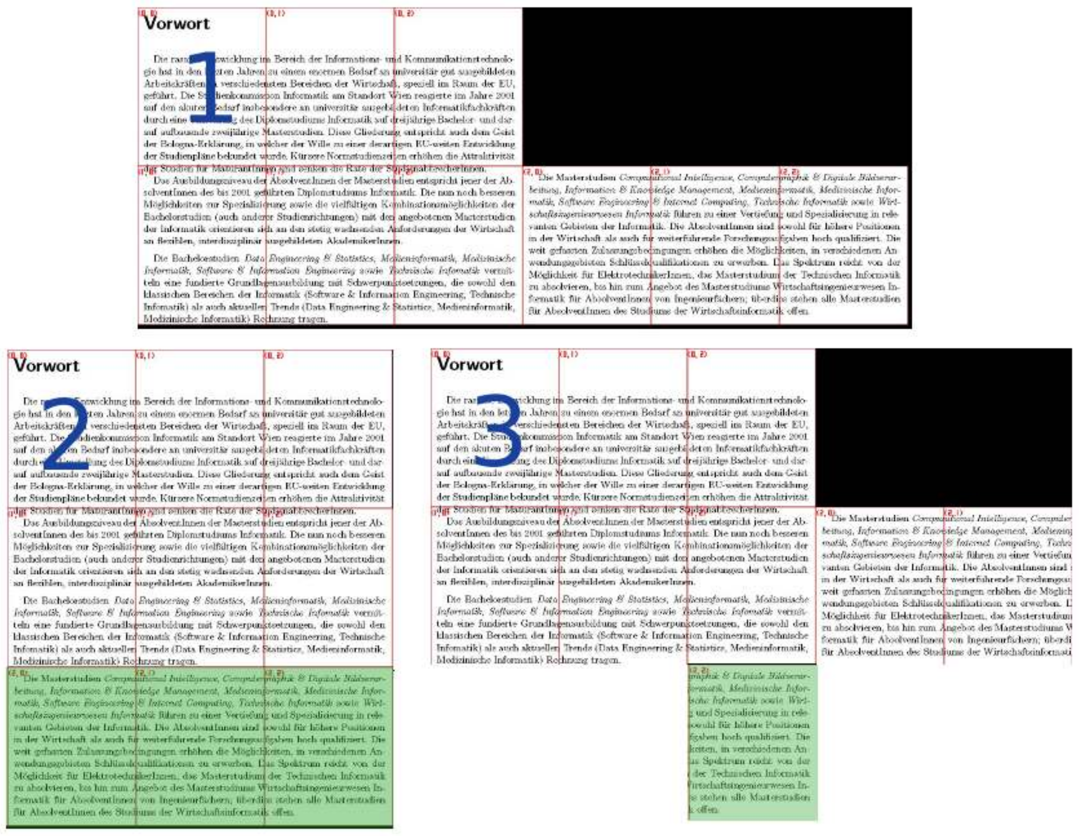


Figure 5.8: In order to move from solution 1 to solution 2 the greedy search would have to pass through solution 3. However, solutions 1 and 3 both have the same number of external edges and so the same score. In this case the transition to the correct result will only be made if the search can see at least 3 moves ahead.

As before, the lookahead requirement becomes quickly intractable. However, since all previous states are recorded, this post-processing step is guaranteed to obtain a final state that's either better or at least equivalent to the one it started with. This guarantee cannot be made for the previous correction heuristic.

5.3.3 Evaluating the error-correcting methods

Both of the above error-correcting mechanism slow down the run-time of the search significantly, as shown in Figure 5.9. The problem is that, even though infinite cycles are detected, the algorithm can still make a large number of moves before it returns to a previous state where a cycle can be stopped. Additionally, the error correcting methods make the runtime of the algorithm a lot more erratic as it ultimately depends on the nature of the encountered cycles. As can be seen in Figure 5.9, while the runtime of the original Prim algorithm increases smoothly with the number of shreds, the error correcting variants have many more peaks and valleys.

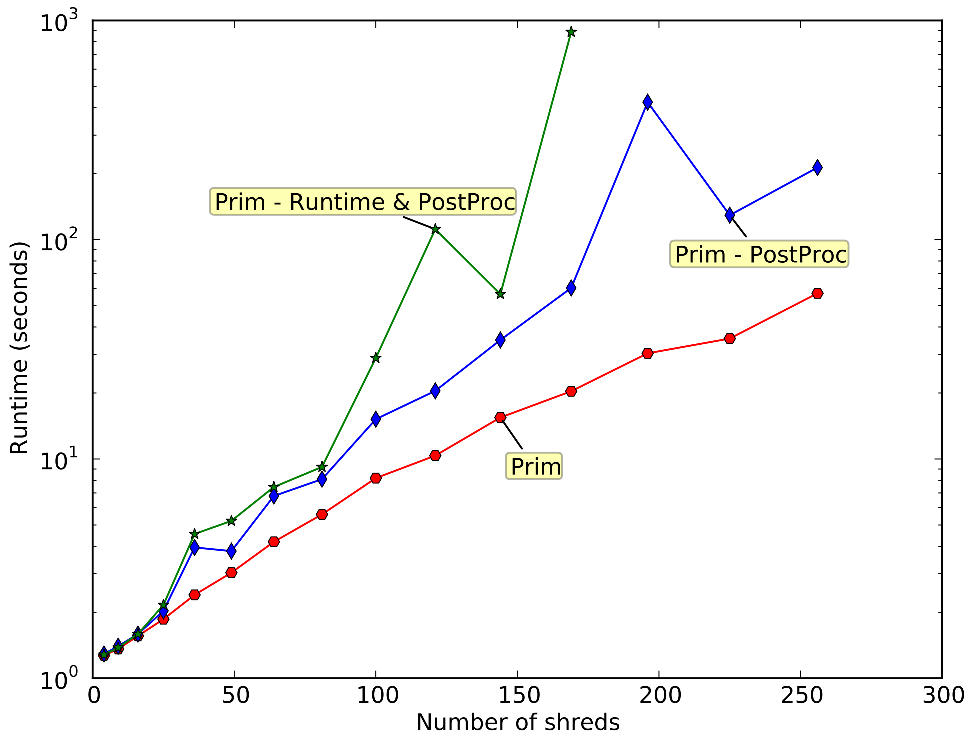


Figure 5.9: Comparison between the basic Prim algorithm and the enhanced versions either with just the post-processing step or with both the run-time corrections and the post-processing. In the case of “Prim - Runtime & PostProc”, the last 3 data points weren’t generated as their run took longer than 20 minutes and was thus stopped.

The performance hit shown above limits the size of the lookahead that can be used and the size of the lookahead can severely limit the performance boost obtained. With a lookahead of 1, the corrections done during the running of the search are inconsistent and may actually hurt the result. The post-processing step however provides a small but consistent boost in performance (see Figure 5.10)

5.4 Modularity

As shown in Section 7.2 and throughout literature (eg: [45, 40, 9]), the reconstruction of cross-cut documents is a very difficult problem. As such, we’ve strived for modularity in all aspects of the proposed solution as to allow for ease of improvement and adaptation at a later time.

The “Kruskal” search function can also allow for modularity, by incorporating a feature called *early stopping*. Since the logic behind the Kruskal method is to, at each step,

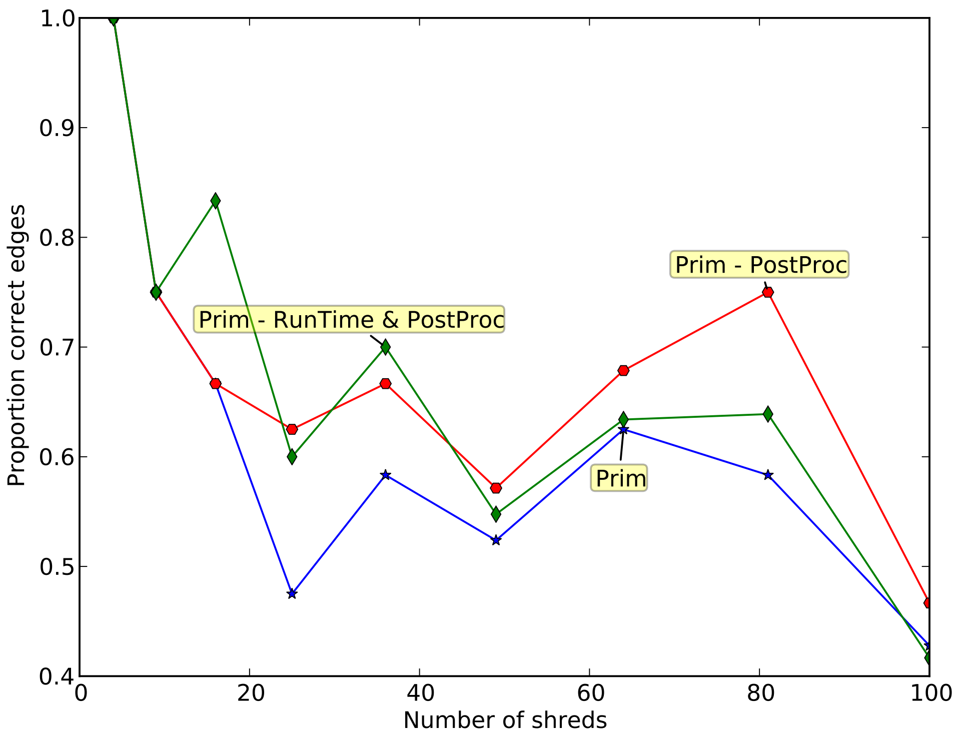
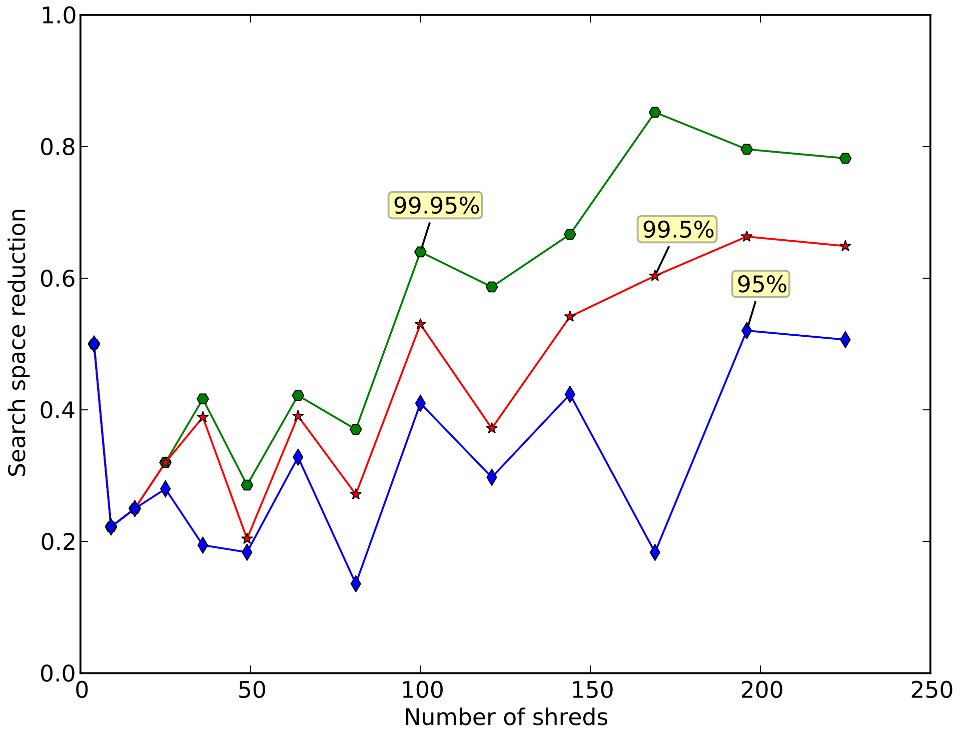


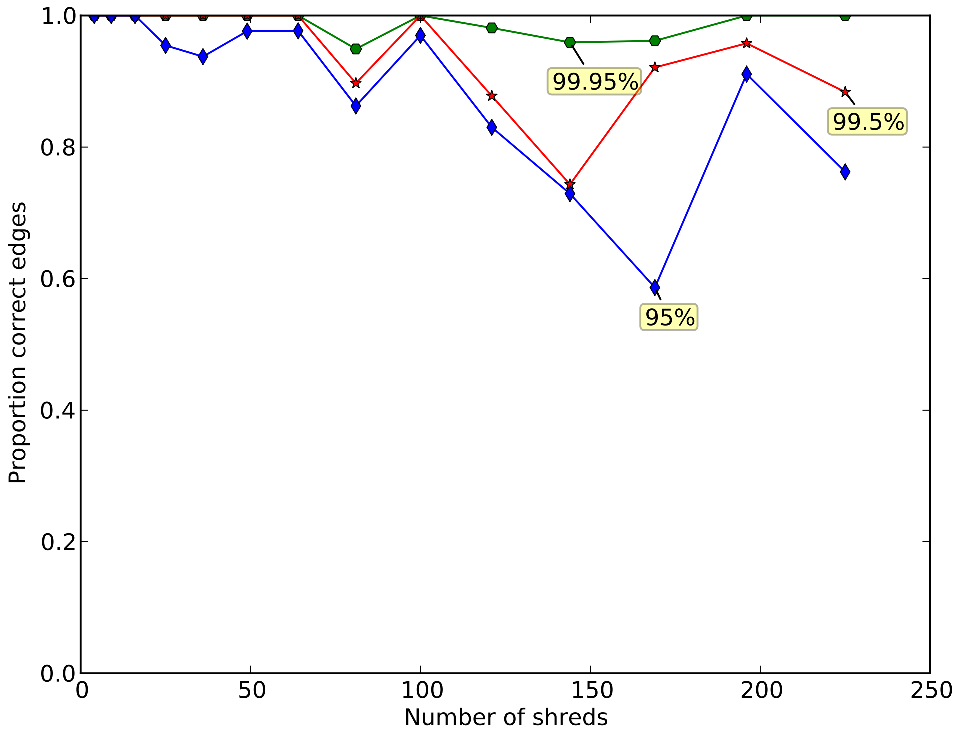
Figure 5.10: Comparison between the basic Prim algorithm and the enhanced versions either with just the post-processing step or with both the run-time corrections and the post-processing. The run-time corrections are not very consistent when using a small lookahead

pick the most likely edge, then we can intuitively return a partial solution by stopping the execution of the search when the most likely edge isn't likely enough. This allows us to use the presented search heuristic as a first step in a larger system, by reducing the search space to a dimension on which a more complex method can function. By varying the stopping condition, we can juggle a trade-off between a more aggressive reduction of the search space and an increased chance of introducing errors into the partial solution, as shown in Figure 5.11

As can be seen, even the extremely conservative *99.95%* stopping condition helps us reduce the search space to between 40% and 80% of its original size. Since the complexity of the search function is exponential in the number of pieces, these reductions are significant and may allow previously intractable search methods to now be run on the reduced search space. As seen from the second graph, greater reductions are possible if we can tolerate some errors in the output.



(a) The reduction in search space corresponding to the 3 stopping conditions. “Search space reduction” is defined as $\frac{\text{Final no. pieces}}{\text{Initial no. pieces}}$.



(b) The proportion of errors incurred by the 3 stopping conditions.

Figure 5.11: Trade-off between reduction in search space and errors incurred by 3 different stopping conditions.

Chapter 6

Pre-Processing

6.1 Motivation

As previously discussed in Section 2.2, a pre-processing step is employed to remove some of the noise introduced by the shredding and scanning process and bring the shreds to a standard form which can be used in the rest of the algorithm. To shortly re-iterate, we want to take the scanned image and return individual images for each shred, such that the shreds are correctly orientated, are all the same size and contain only white and black pixels.

The goal of the pre-processing step can be intuitively seen in Figure 6.1.

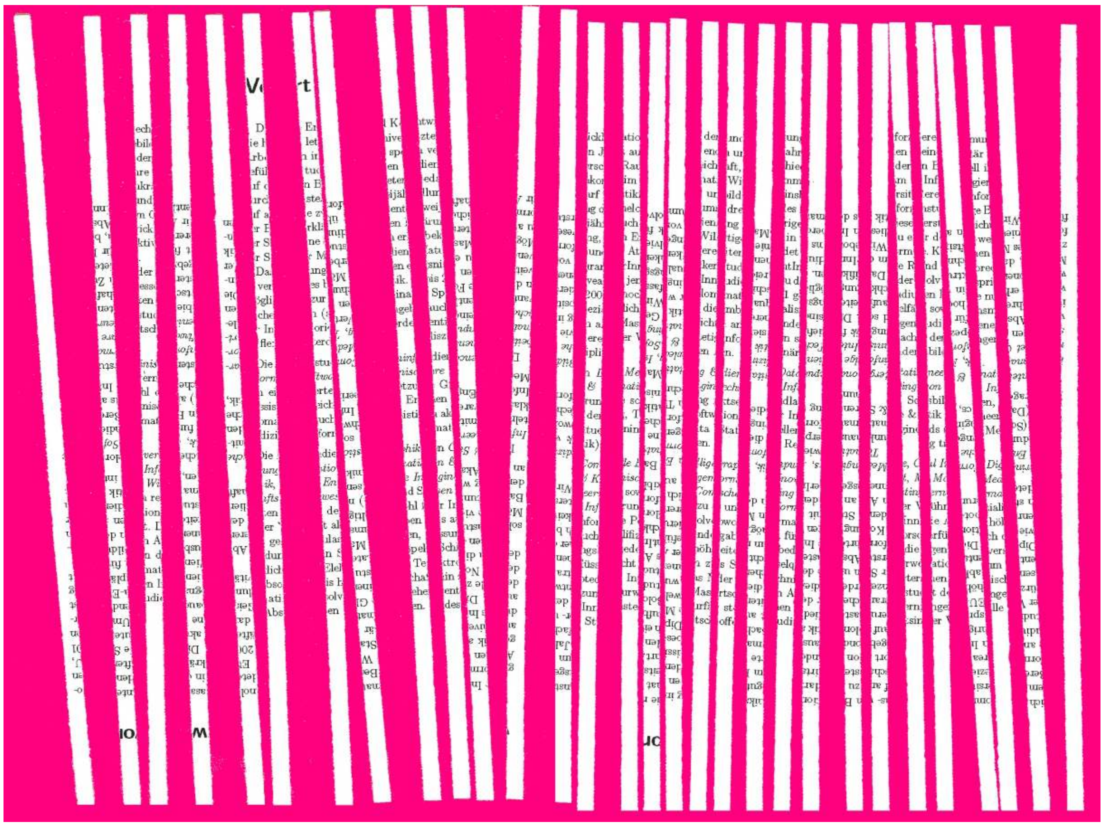
6.2 Segmentation and Skew

First of all, the shreds need to be picked out from the background. In order to simplify this process, a background colour which is significantly different from both white and black needs to be chosen (in the case of Figure 6.1 it has an RGB value of *(255,127,0)*). In order to identify the shred pixels, we can declare a threshold value τ and then say that a pixel p belongs to the background if [1]

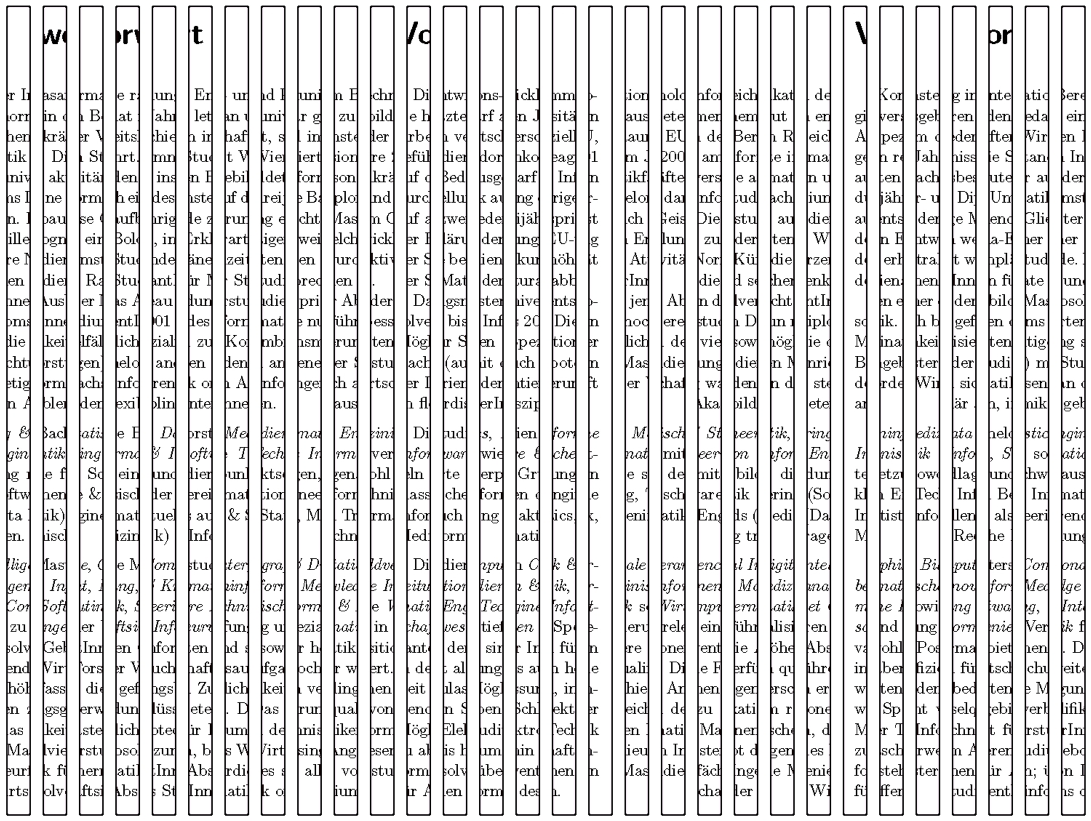
$$|p[0] - background[0]| + |p[1] - background[1]| + |p[2] - background[2]| \leq \tau$$

As reported in [43], automatically setting a τ value is not easy, as the optimal value depends on the noise level in the scanned image. We just choose a value empirically, by running a few tests on the input data.

[1]here we assume that $p[0]$, $p[1]$ and $p[2]$ correspond to the red, green and blue values of pixel p



(a) Scanned image given as input.



(b) Individual *ideal shred* images outputted.

Figure 6.1: Desired input and output of the Pre-Processing function.

After the potential shred pixels are identified, we next group them into continuous blobs. This is done by assigning an initial set to each pixel and then checking each pixel's 8 neighbours. If a neighbouring pixel is found, then the two have their sets merged. After these blobs are identified, the groups smaller than 100 pixels are discarded as noise (unless we are working with micro-shredders, it's quite unlikely a shred will have a size of less than 10x10 pixels). The remaining blobs of pixels are our shreds.

The next step is to fix the skew of the shreds, so that their edges become placed either vertically or horizontally. First the corners of the shreds are identified as the topmost, bottommost, leftmost and rightmost points in each shred. We then identify the edges running between these points, using the criteria that edge pixels are neighbours with at least one background pixel. Once the edges are known, we pick one of the long edges and, for each pixel on that edge, we calculate how much the shred would have to be rotated such that the edge pixel would be placed on a vertical line containing both its corner pixels (see Figure 6.2). Finally, we choose the median of all the calculated angles as the final angle by which the shred will be rotated.

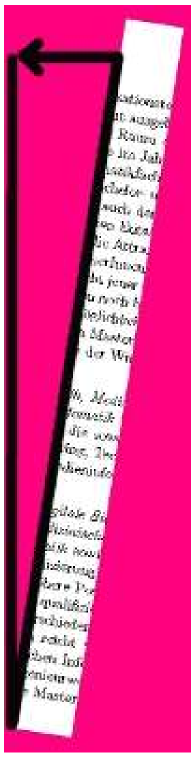


Figure 6.2: The triangle showing how much a certain pixel would need to be moved to be placed on the vertical line (in this case, about 10 degrees).

Once the rotation is known, we can proceed with the segmentation. First a rough bounding box is made around each shred, thus allowing a different image to be created for every shred. Each of these images is then rotated by the required amount as to fix the skew of the shreds. The next step is to go over each pixel once more and record the height and width of each shred for every row and every column in the shred. The median height and median width over all these values are then taken as the final dimensions of the shreds and each image is cropped to that size.

We now have individual images for each shred, which are all of the same size and which are either correctly oriented or rotated by 180 degrees.

6.3 Up/down orientation detection

This is the final step which will make sure that all shreds are “the right way up”. Here we want to identify the shreds which are upside down and rotate them by 180 degrees.

We employ a simple method based on row detection. The goal is to detect both the “inner” and “outer” rows on each shred (see Figure 6.3). The outer rows are defined as those y coordinates in which a transition is made between a row that contains no black pixels and a row that contains at least 1 black pixel. The resulting “outer rows” are then filtered such that only those wider than a minimum threshold are kept, since a row that is only 1 or 2 pixels wide is quite likely spurious. We then want to find an “inner row” inside each remaining “outer row”. The inner rows are defined as the y coordinates exhibiting the greatest difference between the sum of black pixels in two consecutive rows. So when we detect the greatest increase in number of black pixels between 2 consecutive rows, we call this the upper limit of the inner row, and when we detect the greatest decrease we call it the lower limit of the inner row.

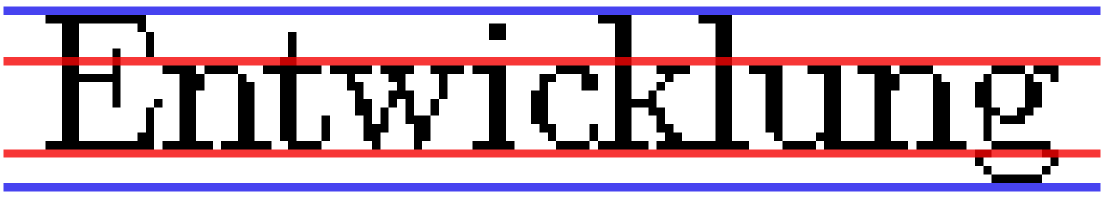


Figure 6.3: Red lines delimitate the inner row, blue lines delimitate the outer row.

After this is done, we count the number of black pixels in the “upper region” and “lower region”, which are the areas between the inner and outer rows placed below and above each shred (see Figure 6.4). We then sum up all of these numbers and predict that a shred will have more black pixels in the “upper regions”.

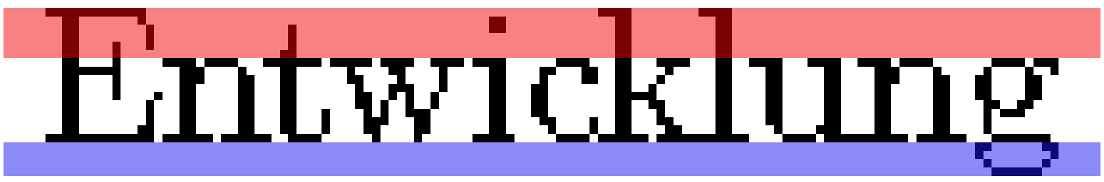


Figure 6.4: The upper region is in red and the lower one in blue. For Roman script, the upper regions will generally contain more black pixels.

This simple orientation detection method does surprisingly well[2], as can be seen in Figure 6.5.

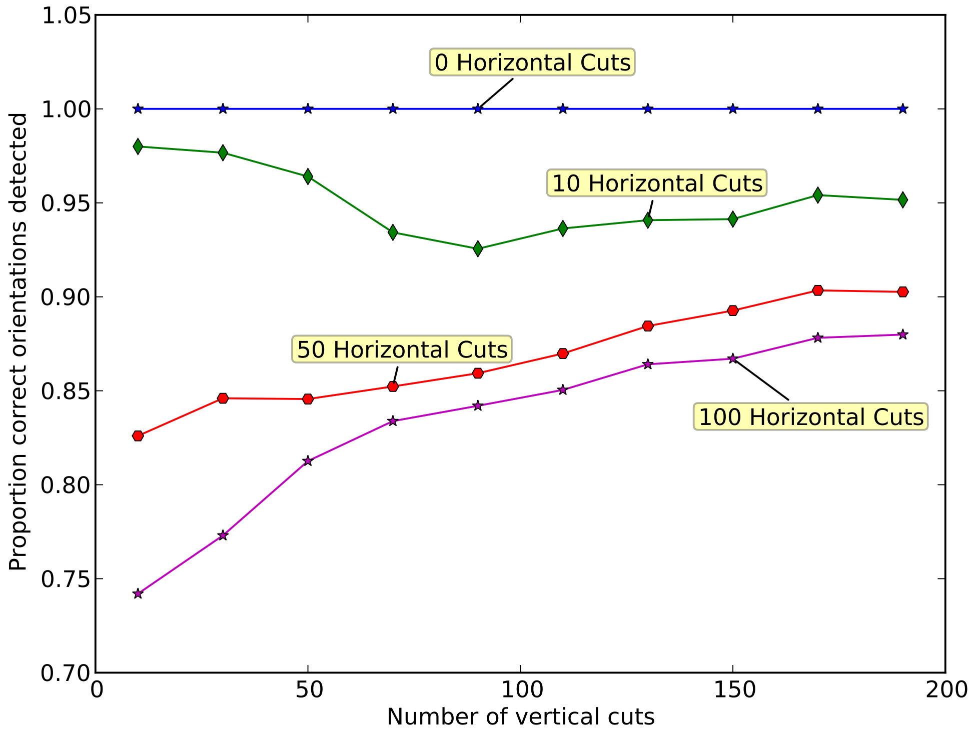


Figure 6.5: The proportion of correctly oriented shreds.

As can be seen, the results are perfect for strip-cut documents (i.e. 0 horizontal cuts) and steadily degrade as we introduce more horizontal cuts. This happens because horizontal cuts reduce the number of rows printed on each shred and therefore make the system more vulnerable to noise. It is worth noting that in the lower two curves, the odd behaviour of the performance getting better as the number of vertical cuts increases is caused by completely white pieces. Completely white pieces are declared as being always correctly oriented and, when both the number of vertical and horizontal cuts are high, there are a lot of completely white pieces. For the lower two curves, increasing the number of vertical cuts increases the number of white pieces faster than the added noise can degrade the performance, so the overall proportion of correctly predicted orientations increases.

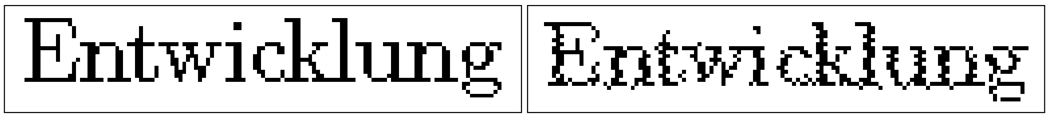
In conclusion, this simple orientation detection method works quite well as long as the number of horizontal cuts is small. For more robust solutions that handle cross-cut documents better or that also work on non-Roman script, one of the methods discussed in Section 3.4 could be implemented.

[2]This experiment uses ideal shreds. Preliminary tests indicate that the method works similarly on real scanned shreds, as long as the skew correction function produces results within 1-2 degrees of vertical.

6.4 Limitations

There are two main limitations of the pre-processing method described above which together were severe enough to prevent the system from working on real scanned shreds.

The first problem comes from the skew correction. It seems that any rotation[3] above 1 or 2 degrees can significantly distort the source image (see Figure 6.6). As shown in Section 4.3.3, this type of noise will deteriorate the performance of the probabilistic model. The problem here is exacerbated since not all shreds will be rotated the same amount and therefore some shreds will be distorted more than other. This non-uniformity further confuses the scoring function.



(a) Original document. (b) Document rotated by 10 degrees.

Figure 6.6: The noise introduced by rotating a document and recasting it to binary data.

The second problem is caused by the "curviness" of the strips which was briefly discussed in Section 3.4. While at a casual glance the original scanned strips appear to be straight, after pre-processing them it becomes clear they are actually curvy (see Figure 6.7). This is a big problem for our scoring function, since it only looks at a thin context on the margin of the shred, and this is exactly the area affected by this issue.

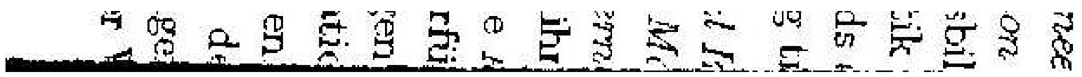


Figure 6.7: A section of a real processed shred. The curviness causes the segmentation to fail and therefore makes part of the margin completely black. This seriously affects the performance of the scoring function.

One solution for the above problems would be to increase the complexity of the methods employed by the pre-processing function. The rotation noise could be mitigated by doing the rotation at a high resolution and then downsampling the image. Specific smoothing operators could also be employed as to minimize the difference between shreds rotated by different amounts. The issue with curvy strips could could be mitigated by the method suggested in [43], namely fitting a polynomial in order to find the true edges. Afterwards the shred could be forced back into a rectangle, though the noise introduced by this modification might further complicate the task.

Alternatively, another solution would be to improve the scanning process as to remove these problems before the pre-processing step. This approach is taken in [8], where the authors find that carefully securing the strips to a rigid background can eliminate both their tendency to curve and the need to correct their skew by more than a couple degrees.

[3]Document rotation was done using both the Python Imaging Library [27] and the ImageMagick "convert" command [24]. No qualitative difference was observed between the two methods.

Chapter 7

Conclusions

7.1 Overview

This project looks at the problem of reconstructing shredded black and white documents, starting from a scanned image of the shreds and ending with a total or partial reconstruction of the original document. The reconstruction is split into 3 functions, “Pre-process”, “Score” and “Search”, each of which is analysed individually.

As part of pre-processing, we define the notion of an *ideal shred* and discuss methods by which such shreds may be extracted from a real scanned image. In particular a new shred orientation method is discussed, which is computationally efficient and works well on strip-cut documents.

The presented scoring function represents a departure from all other scoring/costs functions previously presented in literature. By adopting a probabilistic framework, we improve upon the performance of the previous state-of-the-art method while also getting rid of the ad-hoc heuristics the previous method had used to deal with issues such as white-on-white matches. Additionally, our proposed function is shown to be more robust on most noisy documents and is also more easily composed with other scoring functions, as demonstrated by its combination with “RowScore”.

We analysed the search heuristics used so far and proposed a novel “Kruskal inspired” variant. The Kruskal heuristic outperforms the previously proposed methods and also easily lends itself to compositions with other search methods, via its “early stopping” functionality. Finally, we analyzed a common shortcoming of the heuristics, namely a “cascading” effect and tested out some possible solutions for this problem.

7.2 Results

A major difference can be observed between the results obtained on the strip-cut and cross-cut variants, as shown in Figure[1] 7.1.

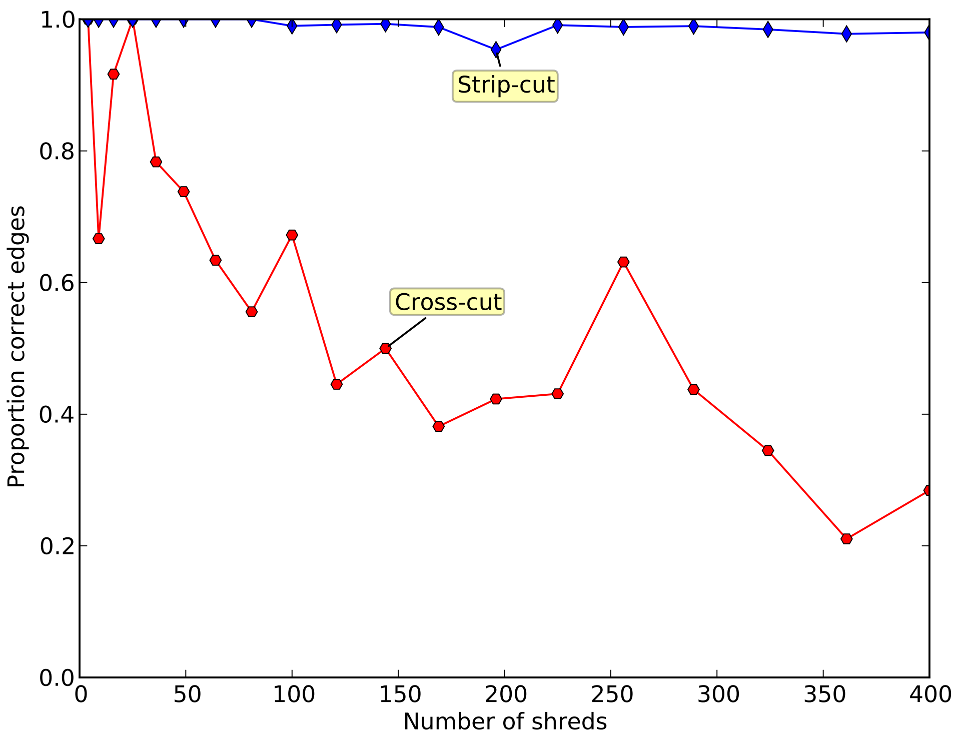
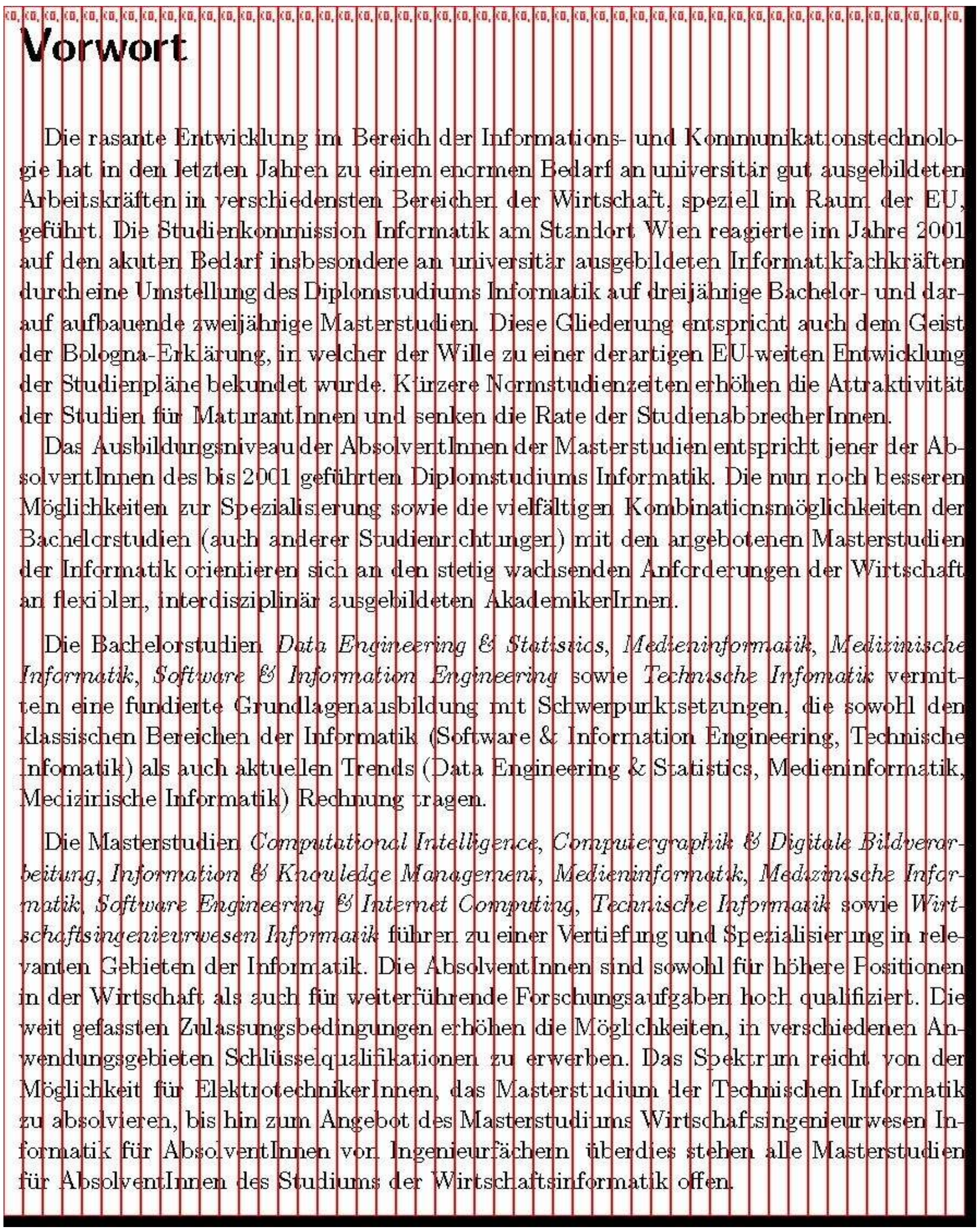


Figure 7.1: The performance of the system on strip-cut and cross-cut documents. While the strip-cut variant is almost completely solved, the cross-cut one proves to be significantly more difficult.

In order to get a better idea of what these numbers mean, we can take a look at some sample reconstructed documents. A document cut into 49 strips is shown in Figure 7.2 and a document cross-cut into 49 rectangles is shown in Figure 7.3.

We can see that with a 64% completion rate the reconstructed cross-cut document has many readable sections. Figure 7.4 shows all the subgroups of the document that were correctly reconstructed.

[1]These results are obtained using virtually shredded documents, the Prim search heuristic and the probabilistic scoring function



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.

Figure 7.2: As expected, the strip-cut document is perfectly reconstructed.

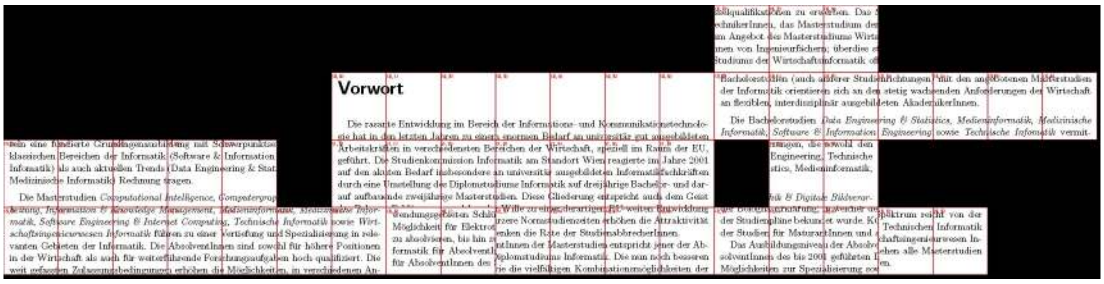
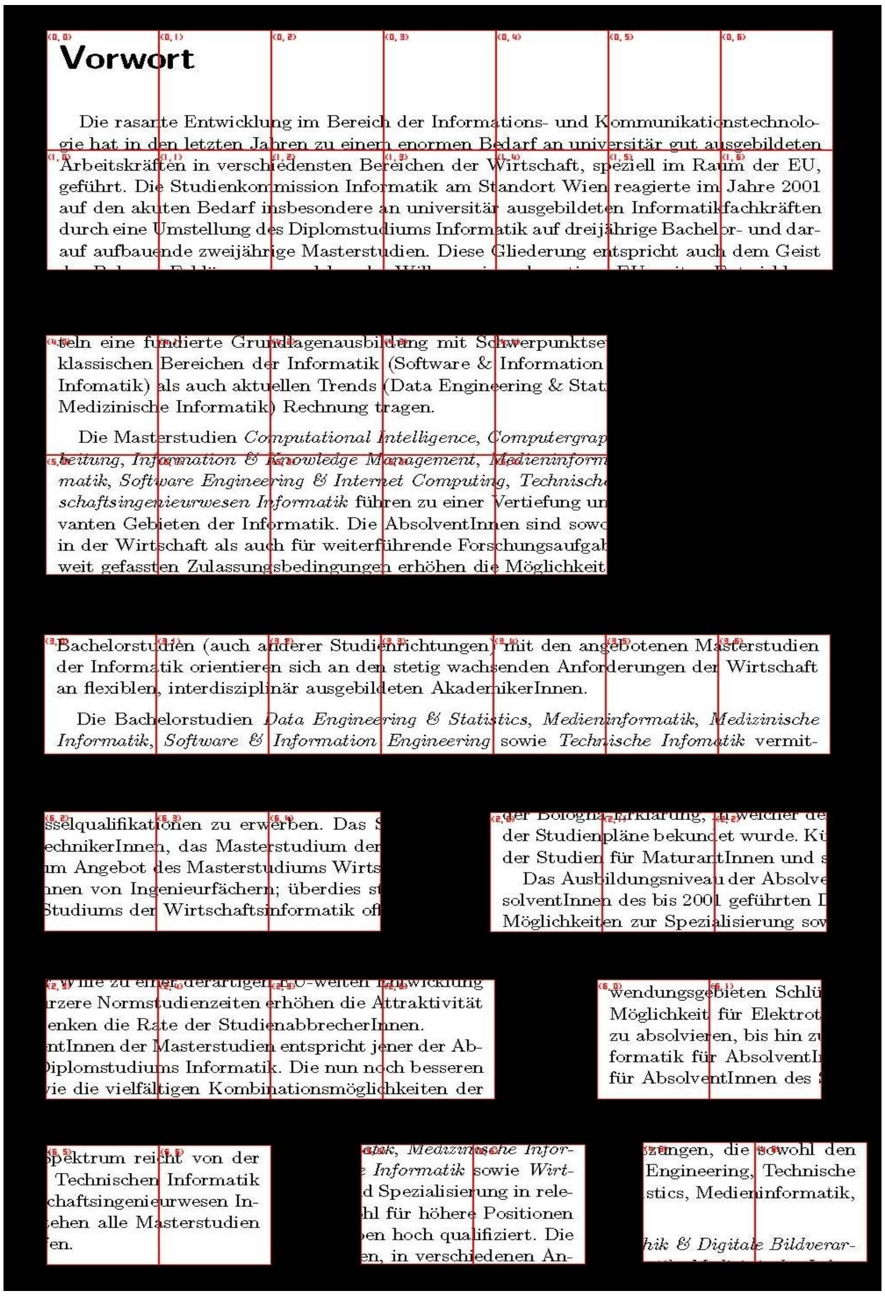


Figure 7.3: This is the outputted document for a 7x7 cross-cut. The edges in this document are 64% correct.



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

teln eine fundierte Grundlagenausbildung mit Schwerpunktse
klassischen Bereichen der Informatik (Software & Information
Infomatik) als auch aktuellen Trends (Data Engineering & Stat
Medizinische Informatik) Rechnung tragen.

Die Masterstudien *Computational Intelligence, Computergrap*
beitung, Information & Knowledge Management, Medieninform
matik, Software Engineering & Internet Computing, Technische
schaftsingenieurwesen Informatik führen zu einer Vertiefung un
vanten Gebieten der Informatik. Die AbsolventInnen sind sowo
in der Wirtschaft als auch für weiterführende Forschungsaufgab
weit gefassten Zulassungsbedingungen erhöhen die Möglichkeit

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* vermit-

sselqualifikationen zu erwerben. Das S
echnikerInnen, das Masterstudium der
ım Angebot des Masterstudiums Wirts
nnen von Ingenieurfächern; überdies st
Studiums der Wirtschaftsinformatik of

der Bologna-Erklärung, in welcher de
der Studienpläne bekundet wurde. Kü
der Studien für MaturantInnen und s
Das Ausbildungsniveau der Absolve
solventInnen des bis 2001 geführten D
Möglichkeiten zur Spezialisierung sov

Wille zu einer derartigen EU-weiten Entwicklung
ırzere Normstudienzeiten erhöhen die Attraktivität
enken die Rate der StudienabbrecherInnen.
ntInnen der Masterstudien entspricht jener der Ab-
Diplomstudiums Informatik. Die nun noch besseren
vie die vielfältigen Kombinationsmöglichkeiten der

wendungsgebieten Schlü
Möglichkeit für Elektrot
zu absolvieren, bis hin zu
formatik für AbsolventIı
für AbsolventInnen des

Spektrum reicht von der
Technischen Informatik
chaftsingenieurwesen In-
ehen alle Masterstudien
en.

atik, Medizinische Infor-
e Informatik sowie *Wirt-*
d Spezialisierung in rele-
hl für höhere Positionen
en hoch qualifiziert. Die
en, in verschiedenen An-

zungen, die sowohl den
Engineering, Technische
stics, Medieninformatik,

hik & Digitale Bildverar-

Figure 7.4: All correctly reconstructed groups of shreds from the cross-cut document. These groups were manually extracted from the reconstructed document.

Here, if we think in terms of the evaluation metric introduced in Section 5.2.3, we can see that the search space has been reduced from 49 shreds to 10 shreds. Even though this only corresponds to 64% correct edges in the final image, the reduction in search space means that, while the initial task would have been somewhat daunting to a human, the new task can be solved with relative ease. Using the "early stopping" mechanism presented in Section 5.4 further simplifies the human's task by returning just the groups the algorithm is certain about. The identified groups with a threshold of 99.5% are shown in Figure 7.5.

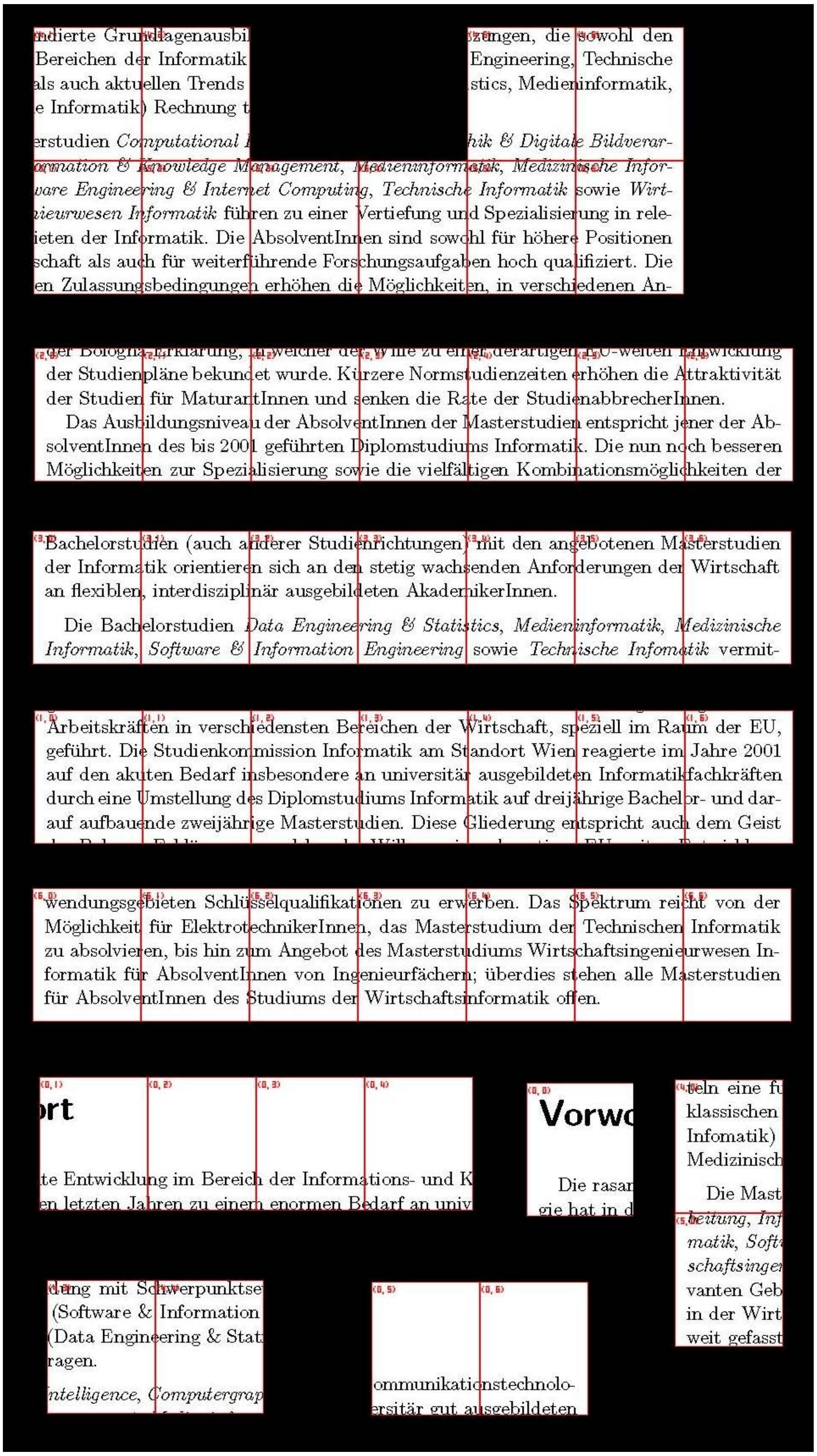


Figure 7.5: The groups automatically identified by the Kruskal search with a stopping threshold of 99.5%.

7.3 Future Work

The post-processing step is in most dire need of work, since it isn't currently usable without spending a large amount of time and effort preparing the shreds before scanning. First of all, the problem with the "curviness" of the strips could be managed by implementing a more complex segmentation method, such as the polynomial fitting or the Hough transform proposed in [43]. Methods of better managing the noise inherent in the shredding, scanning and processing of shreds are also necessary. If the strips are to be segmented and rotated at a high-resolution and then downsampled, the effect of different downsampling and denoising methods needs to be analysed. Lastly, more robust orientation detection methods are needed for small cross-cut documents. Some variants that could be tested include those presented in [10] and [3].

The scoring function, at this stage, can best be improved by being composed with higher-level probabilistic scoring functions. One very simple such function, "RowScore", was tested, but more complex variants could be incorporated. One idea would be to look at the size of the blobs of continuous black pixels formed by a new edge (i.e. the letters situated on the edge). Another method, proposed in [33] would be to look at the actual shape of the aforementioned black, continuous, blobs and predict the probability that these are real letters. Going further, new methods could be devised that do optical character recognition on the shreds and predict the likelihood of detected n-grams. Going in a different direction, different models could be proposed that expand the problem space to incorporate full-colour documents, or even hand-torn documents.

Further work on the search function should focus on fixing the cascading problem presented in Section 5.3. In order to solve this problem a limited exploration of the search space likely needs to be performed. The best way to perform this exploration is an interesting problem that hasn't been previously looked at. Several other additions could also be implemented as to make the system more robust to real world conditions. For instance, the system should be able to handle double-sided documents, as well as documents that contain both text and pictures. Additionally, the degradation of the system when presented with missing shreds or with shreds that don't belong to the current document should be explored.

Bibliography

[1] A. Amin and S. Fischer. A document skew detection method using the Hough transform. *Pattern Analysis & Applications*, 3(3):243–253, 2000. (Cited on page 21.)

[2] Cook W. Applegate, D. and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003. (Cited on page 19.)

[3] H. Aradhye. A generic method for determining up/down orientation of text in roman and non-roman scripts. *Pattern Recognition*, 38(11):2114–2131, 2005. (Cited on pages 22 and 64.)

[4] B. Avila and R. Lins. A fast orientation and skew detection algorithm for monochromatic document images. In *Proceedings of the 2005 ACM symposium on Document Engineering*, pages 118–126. ACM, 2005. (Cited on page 21.)

[5] D. Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981. (Cited on page 21.)

[6] B. Biesinger. Enhancing an evolutionary algorithm with a solution archive to reconstruct cross cut shredded text documents. *Bachelor's thesis, Vienna University of Technology, Austria*, 2012. (Cited on pages 13 and 17.)

[7] Bhowmick P. Biswas, A. and B. Bhattacharya. Reconstruction of torn documents using contour maps. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 3, pages III–517. IEEE, 2005. (Cited on page 15.)

[8] J. Brassil. Tracing the source of a shredded document. In *Information Hiding*, pages 387–399. Springer, 2003. (Cited on page 58.)

[9] Chakraborty P. Butler, P. and N. Ramakrishan. The deshredder: A visual analytic approach to reconstructing shredded documents. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 113–122. IEEE, 2012. (Cited on pages 16, 21, and 49.)

[10] R. Caprari. Algorithm for text page up/down orientation determination. *Pattern Recognition Letters*, 21(4):311–317, 2000. (Cited on pages 21 and 64.)

[11] DARPA Shredder Challenge. Competition homepage. `http://archive.darpa.mil/shredderchallenge`. (Cited on pages 6 and 7.)

[12] Fleck M. Chung, M. and D. Forsyth. Jigsaw puzzle solver using shape and color. In *Signal Processing Proceedings, 1998. ICSP'98. 1998 Fourth International Conference on*, volume 2, pages 877–880. IEEE, 1998. (Cited on page 15.)

[13] P. De Smet. Reconstruction of ripped-up documents using fragment stack analysis procedures. *Forensic Science International*, 176(2):124–136, 2008. (Cited on page 16.)

[14] Kleber F. Diem, M. and R. Sablatnig. Document analysis applied to fragments: feature set for the reconstruction of torn documents. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 393–400. ACM, 2010. (Cited on pages 16, 21, and 22.)

[15] Birattari M. Dorigo, M. and T. Stutzle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, 2006. (Cited on page 19.)

[16] H. Freeman and L. Garder. Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. *Electronic Computers, IEEE Transactions on*, (2):118–127, 1964. (Cited on page 15.)

[17] T. Geller. DARPA shredder challenge solved. *Communications of the ACM*, 55(8):16–17, 2012. (Cited on page 16.)

[18] Malon C. Goldberg, D. and M. Bern. A global approach to automatic solution of jigsaw puzzles. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, pages 82–87. ACM, 2002. (Cited on page 15.)

[19] Bottou L. Howard-P. Haffner, P. and LeCun Y. Djvu: Analyzing and compressing scanned documents for internet distribution. In *Document Analysis and Recognition*. Springer, 1999. (Cited on page 25.)

[20] Arps R. Chamzas-C. Dellert-D. Duttweiler D. Endoh T. Equitz E. Ono F. Pasco R. Sebestyen I. Starkey C. Urban S. Yamazaki Y. Hampel, H. and T. Yoshida. Technical features of the JBIG standard for progressive bi-level image compression. In *Signal Processing: Image Communication*. Elsevier, 1992. (Cited on page 25.)

[21] D. Heingartner. Back together again. `http://www.nytimes.com/2003/07/17/technology/back-together-again.html`, 2003. (Cited on page 6.)

[22] Paul VC Hough. Machine analysis of bubble chamber pictures. In *International Conference on High Energy Accelerators and Instrumentation*, volume 73, 1959. (Cited on page 22.)

[23] Danishjuyan i Musalman-i Payraw-i Khatt-i Imam. *Documents from the U.S. espionage den : America, supporter of usurpers of the Qods*. Teheran: Muslim Students Following the Line of the Imam, 1980. (Cited on page 5.)

[24] ImageMagick. Command-line processing. `http://www.imagemagick.org/script/command-line-processing.php`. (Cited on page 58.)

[25] Oliveira-L. Justino, E. and C. Freitas. Reconstructing shredded documents through feature matching. *Forensic Science International*, 160(2):140–147, 2006. (Cited on pages 15 and 16.)

[26] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956. (Cited on page 41.)

[27] Python Imaging Library. Library homepage. `http://www.pythonware.com/products/pil/`. (Cited on page 58.)

[28] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997. (Cited on page 19.)

[29] W. Morandell. Evaluation and reconstruction of strip-shredded text documents. *Master's thesis, Vienna University of Technology, Austria*, 2008. (Cited on page 37.)

[30] Drewsen-P. Nielsen, T. and K. Hansen. Solving jigsaw puzzles using image features. *Pattern Recognition Letters*, 29(14):1924–1933, 2008. (Cited on page 15.)

[31] Papaodysseus-C. Exarhos-M. Triantafillou-C. Roussopoulos G. Panagopoulos, T. and P. Roussopoulos. Prehistoric wall-paintings reconstruction using image pattern analysis and curve fitting. *WSEAS Transactions on Electronics*, 1(1):108–13, 2004. (Cited on page 8.)

[32] Panagopoulos-T. Exarhos-M. Triantafillou-C. Fragoulis D. Papaodysseus, C. and C. Doumas. Contour-shape based reconstruction of fragmented, 1600 BC wall paintings. *Signal Processing, IEEE Transactions on*, 50(6):1277–1288, 2002. (Cited on page 8.)

[33] Diem-M. Kleber-F. Perl, J. and R. Sablatnig. Strip shredded document reconstruction using optical character recognition. In *Imaging for Crime Detection and Prevention 2011 (ICDP 2011), 4th International Conference on*, pages 1–6. IET, 2011. (Cited on pages 18, 23, and 64.)

[34] M. Prandtstetter. Two approaches for computing lower bounds on the reconstruction of strip shredded text documents. Technical Report TR18610901, Technishe Universitat Wien, Institut fur Computergraphik und Algorithmen, 2009. (Cited on pages 17, 19, and 37.)

[35] M. Prandtstetter and G. Raidl. Combining forces to reconstruct strip shredded text documents. In *Hybrid Metaheuristics*, pages 175–189. Springer, 2008. (Cited on pages 11, 14, 17, 19, 23, 24, 29, 30, and 37.)

[36] M. Prandtstetter and G. Raidl. Meta-heuristics for reconstructing cross cut shredded text documents. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 349–356. ACM, 2009. (Cited on pages 11, 17, 19, 37, 38, and 45.)

[37] R. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957. (Cited on page 19.)

[38] S. SantoshKumar and B. ShreyamshaKumar. Edge envelope based reconstruction of torn document. In *Proceedings of the Seventh Indian Conference on Computer Vision, Graphics and Image Processing*, pages 391–397. ACM, 2010. (Cited on page 15.)

[39] C. Schauer. Reconstructing cross-cut shredded documents by means of evolutionary algorithms. *Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms*, 2010. (Cited on page 17.)

[40] Prandtstetter-M. Schauer, C. and G. Raidl. A memetic algorithm for reconstructing cross-cut shredded text documents. In *Hybrid Metaheuristics*, pages 103–117. Springer, 2010. (Cited on pages 11, 17, 19, 37, 45, and 49.)

[41] ShreddingMachines. DIN security levels. `http://www.shreddingmachines.co.uk/din32757-1.asp`. (Cited on pages 8 and 9.)

[42] T. Sikora. The MPEG-7 visual standard for content description-an overview. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(6):696–702, 2001. (Cited on page 22.)

[43] A. Skeoch. *An Investigation into Automated Shredded Document Reconstruction using Heuristic Search Algorithms*. PhD thesis, PhD thesis, University of Bath, UK, 2006. (Cited on pages 15, 21, 53, 58, and 64.)

[44] Liston-T. Skoudis, E. *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses (2nd Edition)*. Prentice Hall, 2006. (Cited on page 6.)

[45] Massad-Y. Sleit, A. and Musaddaq M. An alternative clustering approach for reconstructing cross cut shredded text documents. *Telecommunication Systems*, pages 1–11, 2011. (Cited on pages 11, 18, 20, 23, 24, 29, 30, 37, 38, 41, 45, and 49.)

[46] A. Ukovich and G. Ramponi. Features for the reconstruction of shredded notebook paper. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 3, pages III–93. IEEE, 2005. (Cited on page 22.)

[47] Ramponi-G. Doulaverakis-H. Kompatsiaris-Y. Ukovich, A. and M. Strintzis. Shredded document reconstruction using MPEG-7 standard descriptors. In *Signal Processing and Information Technology, 2004. Proceedings of the Fourth IEEE International Symposium on*, pages 334–337. IEEE, 2004. (Cited on page 22.)

[48] Shredded document reconstruction system "Unshredder". Company homepage. `http://www.unshredder.com`. (Cited on page 6.)

[49] Zhang H. Yang-C. Liu-F. Jain A. Vailaya, A. Automatic image orientation detection. *Image Processing, IEEE Transactions on*, 11(7):746–755, 2002. (Cited on page 21.)

[50] US Federal Trade Commission website. How to keep your personal information secure. `http://www.consumer.ftc.gov/articles/0272-how-keep-your-personal-information-secure`, 2012. (Cited on page 6.)

[51] US Federal Trade Commission website. What is identity theft? `http://www.consumer.ftc.gov/media/video-0023-what-identity-theft`, 2012. (Cited on page 6.)

[52] F. Yao and G. Shao. A shape and image merging technique to solve jigsaw puzzles. *Pattern Recognition Letters*, 24(12):1819–1835, 2003. (Cited on page 15.)

[53] Lai J. Zhang, H. and M. Bacher. Hallucination: A mixed-initiative approach for efficient document reconstruction. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012. (Cited on page 16.)