An Investigation into Automated Shredded Document Reconstruction using Heuristic Search Algorithms

Anna Skeoch

BSc (Hons) Mathematics and Computing

2006

An Investigation into Automated Shredded Document Reconstruction using Heuristic Search Algorithms

Submitted by Anna Skeoch

COPYRIGHT

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Batchelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:

This thesis may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signed:

Abstract

Shredded document recovery is currently a relevant issue, due to the increase in the use of paper shredders both in offices and the home, but the problem has been the subject of little research to date. Current methods, and the relationship between this problem and that of automatic jigsaw puzzle solving are investigated, and a solution involving feature extraction and heuristic searching is proposed, developed and tested. Further investigation is conducted to include reconstruction of double-sided documents, documents with strips missing, and reconstructing multiple documents simultaneously. Evaluation of the methods is conducted, and suggestions put forward for future investigation.

The conclusions reached are that while the method works well on synthetic images, the feature extraction is not robust enough to provide correct solutions when dealing with real images.

Acknowledgements

I would like to thank Dr. John Collomosse for all of his help and advice, and A and M Skeoch for general support.

Contents

1 Introduction **1**

2 Literature Survey **3**

2.1 Existing research into similar problems 3

2.1.1 Shredded document recovery 3

2.1.2 Automatic Jigsaw puzzle solving 4

2.2 Image Processing 5

2.2.1 Extracting the strips 5

2.2.2 Evaluating the suggested solutions 10

2.3 Search methods 11

2.3.1 Heuristics 12

2.3.2 Comparing Search techniques 16

2.4 Conclusions 17

3 Design Specification **19**

3.1 Software Process Model 19

3.2 User Interaction 19

3.3 Programming language 20

3.4 Functional Requirements 20

3.4.1 Input requirements 20

3.4.2 Processing Requirements 21

3.4.3 Output Requirements 22

3.5 Non-Functional Requirements 22

3.6 Evaluation of Output 22

3.7 Project Plan 22

4 Shredded Strip Extraction **24**
4.1 Introduction . . . 24
4.2 Initial Extraction . . . 24
4.2.1 Scanning in a Document . . . 24
4.2.2 Thresholding . . . 25
4.3 Extracting Rectangular Strips . . . 27
4.3.1 Extracting the Rectangle . . . 31
4.4 Comparison of the methods . . . 33
4.5 Extracting Curved Strips . . . 33
4.6 General Comparisons . . . 37
4.7 Conclusions . . . 38

5 Fitness Evaluation **39**
5.1 Introduction . . . 39
5.2 Proposed Methods . . . 39
5.2.1 Pixel Comparison . . . 40
5.2.2 Scale Space Comparison . . . 43
5.3 Reducing Noise . . . 49
5.4 Other Possible Methods . . . 51
5.4.1 Colour Histograms . . . 51
5.4.2 Edge Detection . . . 51
5.5 Conclusions . . . 53

6 Document Reconstruction **55**
6.1 Introduction . . . 55
6.2 Initial Genetic Algorithm . . . 55
6.2.1 Solution Representation . . . 55
6.2.2 Initial Population . . . 56
6.2.3 Selection . . . 56
6.2.4 Crossover . . . 56
6.2.5 Intelligent Crossover Methods . . . 58
6.2.6 Mutation . . . 60
6.3 Testing the Initial System . . . 60
6.3.1 Missing Strips . . . 70

6.4 Extensions to the Initial System 70
6.4.1 Double Sided strips 70
6.4.2 Multiple Documents 75

7 Conclusions **82**
7.1 Overall Conclusions on the System 82
7.2 Conclusions on Implementation and Project Planning 83
7.3 Improvements to the System 83
7.4 General Conclusions on the Design 84
7.5 Future Work 84

A Testing Images **90**

B Project Plan **98**

C Program Code **100**

Chapter 1

Introduction

The ability to destroy hard (or paper) copies of documents has become particularly relevant in the past few years, with the increase in identity fraud, and large scandals such as the Enron accountancy case in 2001. Paper shredders can be easily obtained from any stationary store, and many people use them to destroy their important documents, as do most corporations.

In some cases however, shredders can also be used to destroy potentially incriminating evidence, and it can be an essential job of police and forensic services to recover lost documents. The ability to reconstruct shredded documents could therefore be extremely helpful to the authorities, if it is possible to recover all or most of the shreds.

This problem of shredded document recovery can be extremely time consuming or even impossible when done by hand, essentially piecing together an extended jigsaw puzzle, with the added difficulty of all pieces being of identical shape and size and possibly double-sided. There is therefore obvious motivation to automate the process as much as possible, with the potential to increase both accuracy and speed of reconstruction.

While there is clearly demand for these types of services from professional companies, there is currently very little evidence in the literature of academic study into the process. However, some examples do exist, and investigation will be done into the methods used to see if anything useful can be taken from the results.

There are clear parallels between shredded document reconstruction and automatic jigsaw puzzle solving though. This has been an area of study since the mid-60s [5], although the focus has generally been on using the shape of the pieces to reconstruct the puzzle [6], which would be less useful when reconstructing shredded documents. There are some examples of solvers using image content though, which could potentially be a source of information.

There are other issues that should possibly be considered. The problem of

document reconstruction for investigative purposes generally implies trying to regain data – which would most likely be in the form of text. Therefore, it might be possible to utilise this when reconstructing. For example, trying to recognise familiar words or phrases, or just putting letters together correctly. This would most likely mean referring to databases of stored information, which leads to the added problems of slowing down the program, and also of constructing the initial databases.

Generally, the aim of the project is to design and implement some software that will input a scanned image of shredded document strips, and automatically recover the original, outputting the result as another image. It would seem that solving the problem with both speed and accuracy requires a combination of good image processing, to both extract the initial scanned image and judge the accuracy of the reconstruction, and a good search technique to quickly reduce the number of possible solutions.

The project will be broken down into the following stages:

- Researching current methods for image processing and heuristic searching, and previous investigations into reconstructing shredded documents and jigsaw puzzles (discussed in chapter 2).
- Creating an initial design and constructing project requirements, based on the research, and choosing an overall software structure for the project (chapter 3).
- Implementing image processing techniques to extract the relevant sections of a scanned image and output into a data structure (chapter 4).
- Developing a method of testing pieced together images for accuracy (chapter 5).
- Implementing a search algorithm that will search through possible solutions and arrive at an optimal one within a suitable timeframe (chapter 6).
- Testing the completed system, and if possible, extending the basic program to include processing multiple documents simultaneously, and processing two sided documents (chapter 6).
- Evaluation of results of the project, and ideas for future developments (chapter 7).

Chapter 2

Literature Survey

2.1 Existing research into similar problems

2.1.1 Shredded document recovery

The problem of automatic shredded document recovery has been sparsely researched to date, although there are a few recent papers waiting to be published ([1], [2]). Justino *et al* [3] investigate piecing together documents that have been shredded by hand. The main focus of their investigation is piecing together shreds based on their shape, which is clearly less relevant to this project, as machine shredded document strips are essentially the same size and shape.

Ukovich *et al* [4] suggest the use of content-based image retrieval to try and match similar shreds. Content-based image retrieval is a way of sorting images using their own visual content (such as colour or texture), rather than labelling them with descriptive key words. Generally, feature extraction methods are used to do this. Ukovich *et al* use this technique to distinguish between shreds from different documents, as shreds with similar content will most likely belong to the same document. This could be a useful technique to use later in the project when piecing together multiple documents.

In particular they suggest using MPEG-7 descriptors as a method of distinguishing between image shreds. These are a particular type of descriptors developed by the Moving Picture Experts Group (MPEG), designed to describe multimedia content. Ukovich *et al* use three colour descriptors, two texture descriptors and two shape descriptors. They get some good results using the colour descriptors, but disappointing results using texture and shape descriptors.

Using content-based image retrieval is a valid technique for solving this problem – it is essentially feature extraction, which is the main area of

image processing that will be examined in this review. However, it does not seem necessary to go to the lengths of using descriptors to classify different documents at this stage of the project. If a large number of documents were being reconstructed, it might be more useful – therefore this will be considered if the bounds of the project extend past the initial requirements.

2.1.2 Automatic Jigsaw puzzle solving

As reconstructing shredded documents is a special case of automatic jigsaw puzzle solving, it is useful to look into previous attempts to solve this problem as well. This has been an area of study since the mid-60s [5], though often the approaches focus mainly on reconstructing puzzles based on shape alone using shape matching, such as Goldberg *et al* [6]. Like [3], this is not particularly relevant to the work in this project. Goldberg goes into little detail on the searching process, but a simple greedy algorithm is used, which would probably not be suitable in this project as the complexity of it would make this type of algorithm impractical (see Section 2.3).

There are several examples of also using the visual information on the pieces, however, such as Kosiba *et al* [7], and Chung *et al* [8]. They make use of the colour of the pieces to determine the accuracy of a match between two pieces. This is a fairly simplistic technique, and most likely colour information alone will not be enough to match two shreds of a document (especially if the document is text-based). Kosiba notes that when solving jigsaws of 25 pieces or more, increasing the number of features used for matching considerable improved the accuracy of the solution. Chung manages to successfully reconstruct a 54 piece puzzle, which suggests that this is certainly a viable method.

Toyama *et al* [9] propose a method for solving jigsaw puzzles consisting entirely of rectangular pieces, using a genetic algorithm for the search, as well as feature extraction. Using rectangular pieces, all the same size, means that the emphasis shifts from matching the shapes of the pieces more towards good feature extraction and efficient searching. This is very relevant, as it uses both techniques to be investigated in this project. Toyama does simplify the problem somewhat – images are only in black and white (which means that comparison is only in one dimension rather than the three RGB dimensions), and it is assumed that pieces will not have to be rotated. Feature extraction purely consists of matching neighbouring pixel values, and so the main focus is on using the genetic algorithm to obtain an optimal solution.

Toyama manages to correctly reassemble an 8x8-piece puzzle using this technique. The results are encouraging, and show that it is possible to use a heuristic search to get an optimal, or near-optimal solution. However, it

seems sensible to gain as much information from the scanned shreds as possible – extracting colour and feature information should aid the search, and prevent reaching a local rather than global maximum solution. Therefore, we shall now examine the different techniques that are available for feature extraction.

2.2 Image Processing

As there are two main applications of image processing to the project – extracting the strips from scanned images, and evaluation of the image created by putting the strips together – techniques appropriate to both of these tasks shall be evaluated. It is often difficult to choose the most suitable technique purely by examining their advantages and disadvantages on paper, therefore the aim shall be to research as many methods as possible, and then use the development stages of the project to decide on the best to use.

The main text used for researching these techniques was Nixon and Aguardo [10], which provides a good overview of many relevant methods. Also used was Seul *et al* [11], which gives a more algorithmic description and explanation of the different techniques, and Russ [12], which provides a slightly different description of some of the methods mentioned here.

2.2.1 Extracting the strips

Initial analysis of the issue of extracting the strips of document from a scanned image suggests there are several stages to this: determining the difference between the background of the image and the strips to be extracted, picking out the rectangular shape of the strips and disregarding any noise, then extracting the relevant pixels and rotating or resizing as necessary to produce strips of the same size and shape. These strips can then be combined to form a new image for the next stage of analysis.

Thresholding

One of the most basic image processing techniques mentioned in Nixon and Aguardo, thresholding involves turning an image into a binary image by determining if a pixel value is above or below a certain value, and turning it either black or white as necessary. It is then easy to pick out regions of interest. For example, thresholding could be done for pixel colour values. In the case of this project, thresholding could be useful to extract the strips from the background image — the background chosen should be a distinctive enough colour that the difference between strips and background are fairly pronounced.

The main difficulty would be choosing the correct thresholding value, taking into account the noise that can be produced by scanning an image. A sample could be taken of the background colour, and then all the pixels could be averaged to produce a suitable value to threshold at. However, this would not take into account the distribution of the pixel colour values around the mean — they might all be close to the mean value, or they might be quite widely distributed. Therefore, a better method might include a measure of the standard distribution as well as the average pixel values.

Morphological Operators

These consist of operations to modify regions of shape in binary images. While these techniques can be useful, especially for removing noise from an image, they may not be suitably robust as a way of extracting the boundaries of the strips. They can be used to initially differentiate between document strips and irrelevant noise though.

Generally, they replace the value of a particular pixel with a new value calculated using neighbouring pixels. The simplest operators are erosion and dilation. Erosion is used to eliminate irrelevant pixels from an image, by removing layers of pixels from the boundary of all regions of interest (called ON regions). This is done gradually, so eventually all but the largest shapes in the image have been eroded.

Dilation is the opposite process – it adds a layer of pixels around regions of interest. Once the main shapes have been identified, dilation can be used to increase the size of the regions to their initial shape. This is known as Opening. Erosion can also be used for a basic method of boundary extraction. An image is eroded a few times, and then the difference between an eroded image and the original is computed. This will gave a vague boundary of the shape. The boundaries would not necessarily be the correct shape and size though, particularly if affected by added noise.

Chain coding

This method was originally introduced in the 60s by Freeman [13], as a method of storing the positions between pixels in the boundary of a shape. The idea is to choose an initial pixel and a direction to move in (see Figure 2.1). The nearest pixel in that direction is chosen, and the relevant code for the pixel is recorded. For example, if the nearest pixel were directly above the initial pixel, a value of 0 would be recorded in the coding. This would continue until the entire shape had been encoded. It is up to the user to either choose a 4-way connectivity code, or an 8-way code. The numbers used for the code are arbitrary — as long as the numbers are consistent, they can be whatever the user decides.

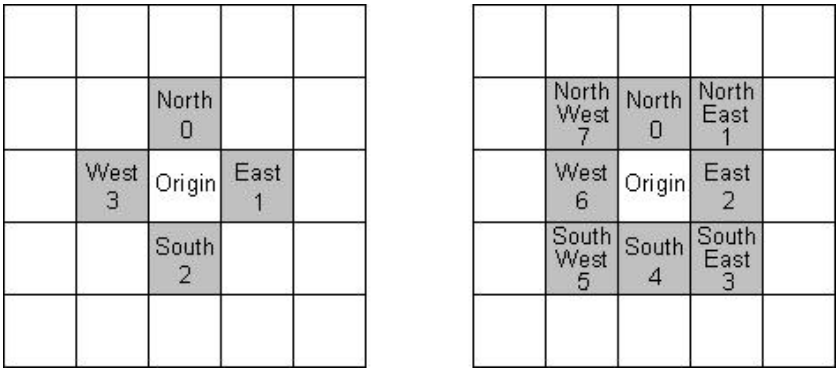


Figure 2.1: Connectivity in Chain Coding (taken from Nixon and Aguardo)

Chain coding is a very simple method of representing the directions of the boundary of a shape. Potentially, it could be used to represent the boundary of a strip, and also determine the orientation of the particular strip — by looking for particular repeated sequences, the edge directions could be calculated. For example, a string of 0s in succession would suggest a line going upwards. This could then be used to rotate the strip to the correct position. However, chain coding is not at all robust to noise — a single pixel out of place would upset the code — or changes in scale and rotation, and does not really provide useful information on shape.

Edge Detection

This is a method of extracting lines from an image, mainly based on looking for changes in contrast, colour or intensity. The idea is that these changes usually occur at the boundaries of a shape. There are two main types of edge detector – first-order operators and second-order operators. First-order operators work by using first order differentiation on an image, and looking for peaks to pick out the edges. There are several versions of first-order operators, including Roberts Cross [14], Sobel [15] and Canny [16].

Second-order operators work on the same principles as first-order, except they use second order differentiation on the image, and look for zero-crossings to find the edges. Versions of these include the Laplacian and the Marr-Hildreth. The more advanced techniques also make use of Gaussian filtering, to reduce the response to noise in the image, which can produce irrelevant or non-existent lines.

Edge detection can potentially be used both when determining the edges of the strips, and also to look for lines in the strips themselves. It is a fairly straightforward technique, and is not too computationally intense, which may become important when considering ways to speed up the final

program. The main issue when using edge detectors is choosing the best type to use. Nixon and Aguardo point out that it is often "difficult to justify the extra complexity associated with the Canny and Marr-Hildrith operators" (pg. 127). Indeed, it would be better to use the simpler techniques where possible, especially if noise can be removed by some methods of filtering.

Corner detection/Image Curvature

Another fairly low-level technique, the idea of corner detection is mainly to look for rapid changes in the direction of edges. The simplest methods just use the change in angle along the length of an edge or curve – taking some connected pixels and looking at the difference in edge direction. This is easy to implement, but can be very vulnerable to noise, and often produces inaccurate results.

More complex methods include approximation to a continuous curve, or fitting a curve to points given by the known position of edges in the image. This reduces errors related to differences in the points of the curve and the positions of the pixels, but is also not particularly robust to noise. Other methods, including the Harris corner detector [17], use autocorrelation. This considers the changes in intensity when a window on a part of the image is moved. This is probably the most reliable technique, but also the most complex.

Corner detection could be used to find the corners of strips, although as it often involves quite a lot of computational effort and unreliable results, other techniques are more likely to be considered first.

Hough Transform

First proposed in 1962, the Hough transform [18] is a versatile technique for finding different shapes in images. The most common use is to find lines and circles or ellipses, but it can be used in a more general form to look for all manner of shapes. The method for line detection works by considering the parametric form of a line:

$$\rho = x \cos\theta + y \sin\theta \tag{2.1}$$

Two parameters are needed to determine the position of the line: θ and ρ, where θ is the angle the line makes with a given axis, and ρ is the length of the line. The Hough transform goes through all possible values of θ and ρ in a given parameter space, and adds up the potential solutions in an accumulator array that is the size of all the possible values of θ and ρ. Once this is complete, it is simply a matter of searching the array for the

maximum values, and these are the most likely θ and ρ values for the correct line passing through the pixels in the image.

For example, all of the possible lines in a given space with a number of pixels would be evaluated by the Hough transform, and any lines that went through any of the points would get additional points in their respective accumulator array places. Then, a search for the maximum would hopefully give only one possible line – the line passing through the maximum number of the points. The principles behind the Hough transform can be extended to other shapes – anything that can be parameterised in an equation. However, the greater the number of parameters, and the greater the dimensionality of the parameter space, the longer the algorithm will take to run.

The Hough transform can also be used to find arbitrary shapes, that may have unknown position, size and orientation. This is known as the Generalised Hough transform (GHT), initially proposed by Ballard in 1981 [19]. The technique uses a similar system, where a shape is defined by a set of parameters and a reference point. However, this method is not very robust to scaling, and also requires a great deal of computation (because of the increased number of parameters), so may not be practical to use in this project.

The more simple form of the algorithm could be used though, both to find the initial rectangles bounding the strips, and to find lines in the strips themselves, which could aid in the reconstruction process. Indeed, Dehong *et al* [20] demonstrate the use of the Hough transform to effectively find aeroplane landing strips in noisy images. The Hough transform is very robust to noise, as it explores all the possible lines that can fit to a set of pixels, rather than using the pixels themselves to suggest possible lines. It is also possible to reduce the computational expense by constraining the range of the θ and ρ values in Equation 2.1 so the smallest possible space is searched. This makes it a very attractive technique to consider.

Active Contours

Active contours, also known as *snakes*, were initially suggested by Kass *et al* [21], as a method of finding the best approximation of the perimeter of an object. They work by placing a snake near to or around a contour of interest in the image. Then, using iteration, the snake moves closer to the actual shape until it fits around it tightly.

The actual approach uses an energy functional that measures how appropriate the contour fit is – the best solution is obtained when the functional reaches a minimum. The contour is parameterised as:

$$v(s) = (x(s), y(s))$$

where x and y are the coordinates of points on the snake, and s ϵ [0,1] is the normalised length around the snake. Then the energy functional is:

$$E_{snake} = \int_{s=0}^{1} E_{curv}(v(s)) + E_{image}(v(s)) + E_{cont}(v(s))ds \tag{2.2}$$

This consists of the sum of several terms which all act on the snake: the internal energy terms — E_{curv}, which forces the contour to be smooth, and E_{cont}, which forces the contour to be continuous; and the external energy term, E_{image}, which attracts the contour towards certain low level features in the image (such as edges). While using snakes could be possible for finding the boundaries of document strips, it might be difficult to guarantee a rectangle, as noise could affect the results.

2.2.2 Evaluating the suggested solutions

Once the strips have been successfully extracted from the scanned image, and combined to form an initial reconstruction of the document, this image must be evaluated to determine the accuracy of the solution. Likewise, when using a search to suggest improvements to the proposed solution, there must be some measure of the new solutions' accuracy. Therefore, features will be extracted in some manner, and used to evaluate the created images.

When deciding techniques to use for evaluation of the suggested solutions, there are two main types to consider: Low level, and high level. Low level techniques, such as edge detectors, tend to extract basic features from an image without considering the underlying shapes — features such as colour, or lines. Higher level techniques, such as snakes, aim to extract the shapes themselves, although often they use low-level features to achieve this.

As previously mentioned, Kosiba *et al* [7] note the colour along the edges of the pieces of a jigsaw, and once they have a tentative match, they compare the colours of the matching edges. If the colours are similar enough, a correct match is assumed. This is a very low level method of feature extraction, but it could be useful in determining if the strips are being matched correctly. However, the results may easily become inaccurate if the colours at the edges of the strips are incorrect — for example, if some of the background colour from the scanned image is still there. Higher level techniques would be more robust to this type of noise.

The Hough transform, or snakes could potentially be used to find shapes in the document contents, especially if there were pictures or graphics included. However, the size of the document strips would most likely mean there would be difficulty picking out any shapes that could be useful, so they would need to be used on the whole images created by combining the strips. The main

problem with implementing this method would be trying to pick out shapes without having any idea what the shapes might be.

Snakes have been used for problems such as face recognition, but faces have definite shape and layout - a round/elliptical shape, with two ellipses near the top, and another ellipse nearer the bottom. As this project aims to reconstruct documents that could contain anything, there can be no guarantees of finding any particular shapes. The best solution to this problem would be to search for general shapes in the images, such as circles/ellipses, rectangles, etc., but not to assume that any will be found. It remains to be seen if this would produce any useful results.

2.3 Search methods

When choosing a search technique, it is important to consider the complexity of the problem space. Assuming that a document contains n strips, the potential number of solutions will be (n!). Allowing for the possibility that a strip may be rotated 180 degrees (as it may be scanned upside down), the number of solutions increases to around 2(n!). This is fairly complex, even for a relatively small number of strips. Chung *et al* [8] point out that this is classed as an NP-complete problem. There are many examples of NP-complete problems, including the Travelling Salesman problem, and the Graph Colouring problem [22].

Generally, searching techniques are divided into two categories: Uninformed (or brute-force), and informed methods. Brute-force methods are guaranteed to find the optimal solution... eventually. Typically, these involve exhaustive searching through a problem space, until the optimum is found. This can be performed in a variety of ways, including working through a list, or working down a tree, either *breadth-first* (going a level at a time), or *depth-first* (going all the way down a branch until the final node is reached).

While these methods do always find the best possible solution, they are often impractical to use, because searching through an entire problem space, especially when it is large, can take an unreasonable amount of time. Therefore, it is generally better to use an informed method, which uses information about the problem space to perform an more intelligent (and hopefully faster) search. Typically, this involves using search heuristics. This will not always find the optimal solution, but will find a good solution in a reasonable amount of time. NP-complete problems always require an exhaustive search to find the best solution, but as this takes so much computation for large problems, these other methods are nearly always used.

There are a wide variety of heuristic methods available, but the focus for this project will be on methods of global optimisation. Once all of the

strips of the document have been extracted, feature extraction will provide an evaluation of how accurate the current configuration of strips is. The purpose of the search is then to maximise this evaluation, by searching through possible alternate configurations in an intelligent way.

2.3.1 Heuristics

Blum and Roli [23], in their overview of heuristics, highlight the distinction between *constructive* (or bottom-up) methods and *local search* (or top-down) methods. Bottom-up algorithms tend to build up solutions bit-by-bit, until a complete solution is returned at the end.

Top-down approaches, on the other hand, begin with a complete solution, and try to improve on it by iteratively trying other possible solutions. It is pointed out that top-down approaches are generally more effective, and indeed, for this project, it seems sensible to view entire solutions before evaluating them, rather than evaluating them a section at a time. Otherwise, it would be easy to erroneously piece together two strips that seem to be adjacent, and then be unable to reach the correct solution.

Hill Climbing

While not strictly considered a heuristic, hill climbing is nevertheless a local search technique and the basis for many other searching methods, and shall therefore be considered for this project. It is essentially a depth-first method of a tree search, that chooses the path it takes based on the closest node to the current solution. Once the new node is evaluated, if it is a better solution than the previous choice, it becomes the new best solution, and the closest node further down the path is chosen as the next node. This continues until there are no more possible moves — it is assumed that the best solution has been reached.

This is based on the idea of climbing a real hill, in that as long as someone keeps moving upwards, they will eventually reach the top. However, the problem with this idea, is that if there are small hilltops that occur before the real hilltop, a person could easily stop there and never reach the true top. Likewise, if there are local maxima in the problem space, the hill climbing method will often get stuck on them and never reach the global maximum.

There are alternatives to the basic hill climbing algorithm that try to improve performance. The Steepest Ascent method (or gradient ascent) evaluates all possible nodes one step further on the path, and chooses the best one from those (similar to climbing the hill by choosing the steepest path to the top). This can still mean getting stuck on local maxima though. Methods

used to avoid this include backtracking to previous states and trying a different route, or random-restart hill climbing, which will try several starting points, running the hill-climbing algorithm for each, and choosing the best result.

Hill climbing is one of the most basic methods of searching, and while it can often be effective in certain circumstances, it is generally considered to be ineffective in more turbulent search spaces (such as those with large numbers of local maxima), when compared to more advanced techniques. There are examples of comparisons in the literature (Mitchell *et al* [24], Talbi and Muntean [25]) which conclude that while hill climbing is the fastest method, the results are generally worse than other potential methods. However, Ackley [26] concludes in his comparison of hill climbing techniques, Genetic Algorithms (GAs) and Simulated Annealing (SA), that a modified version of hill climbing , called SIGH (Stochastic Iterated Genetic Hillclimbing) can give comparable results to both GAs and SA. This technique could therefore be considered for this project.

However, when considering such comparisons, it is important to consider why the comparison is being performed. In the case of Mitchell *et al*, the object of the paper is to demonstrate the advantages of GAs, so naturally the emphasis is on the failing of other techniques. Likewise, Ackley has developed SIGH himself, and therefore wishes to highlight the areas where it can be compared favourably to GAs or SA. Any comparisons which are done purely for the purposes of finding a good technique in particular circumstances will be considered more useful than papers which aim to advocate any technique in particular.

Simulated Annealing

An extension of the hill climbing algorithm, simulated annealing was first suggested as a search technique in the 80s by Kirkpatrick *et al* [27], and independently, by Cerny [28]. Like hill climbing, it is based on a real technique, the physical process of annealing. This involves heating up a solid object until it melts, then cooling it down slowly until it crystallises in the best possible structure. The optimal structure can be obtained when the energy of the solid is at a minimum. The process must be slow, as this allows atoms to move around, and find configurations with lower energy.

Simulated annealing aims to imitate this process, by choosing an initial random configuration, and bringing it to the minimum *energy* state. This is done by selecting a parameter called the *temperature*, or control parameter, which gradually lowers over time, and using it to decide the probability of transitioning from one state to another. Other solutions are considered, and chosen depending on their optimality, but also depending on the current temperature value.

So, less optimal solutions may be considered early on, when the temperature is high, but as the process carries on, only the best solutions will be chosen, eventually reaching the global optimum. The idea is that this will allow more exploration of the problem space early on, avoiding convergence on local maxima.

Blum and Roli point out the importance of choosing an appropriate cooling schedule, in order for the algorithm to perform most effectively. While simulated annealing is an attractive technique to consider, as it is fairly simple to implement and run, it will be necessary to select the temperature value carefully. This seems to be the critical parameter in this algorithm, so may require some fine tuning when put into practice.

Tabu Search

Not strictly a search method in itself, but classed as a metaheuristic (by Glover in 1986), the Tabu Search was first proposed by Glover and Laguna [29]. It was designed to be used in conjunction with other search methods, in order to reach an optimum more successfully. A record is kept of all moves that have been made in a particular problem space, and movement to another node is allowed or prohibited, depending on how recently they were visited, and possibly other criteria.

The idea is that by avoiding nodes that have been frequently examined, greater movement around the problem space will be encouraged, and therefore more solutions will be considered. This lowers the chance of ending on a local maximum. However, avoiding certain nodes can mean ignoring some of the best solutions, so often aspiration criteria are included, which allow the tabu state to be overridden. Generally, solutions which are better than the best current solution will be allowed.

Tabu searches have been effectively used to solve many optimization problems (for examples, see [30]). However, it would significantly increase the amount of computation necessary when conducting a search, and the memory needed might become impractical for this project when looking at a large problem space. For searching a smaller space, it might be possible to implement, but computationally it would probably be more efficient to do several individual searches of the space and choose the best solution.

Genetic Algorithms

Another technique based on an idea from nature — evolution. Genetic algorithms (or GAs) are a sub-section of evolutionary algorithms, that were first proposed by John Holland in the 70s [31]. They remained only a theoretical idea until the 90s, when more powerful computers meant that practical

applications could be attempted. An excellent overview of the most recent developments is given in Reeves and Rowe [33].

The method is based on the principles of natural selection. A particular solution to the problem is represented by a structure, called the *genome*. The structure can be anything, but is most commonly represented as a string of binary numbers, called the *genes*. The initial *population* is composed of a number of randomly created genomes, although this population can be slightly skewed towards good solutions if necessary.

Each genome is then evaluated according to some *fitness function*, and solutions are selected that will be used to create the next generation. Selection tends to include the better solutions that have been evaluated, but normally, poorer solutions are also permitted, so convergence to a local maximum is avoided. Once the parent genomes have been chosen, they are combined to create new child genomes. This is called *crossover*, and usually consists of mixing the genomes of the parents.

Once the children have been created, they are *mutated* randomly, generally by flipping individual genes. Mutation does not always happen, but it introduces an element of extra variety into the method, which also encourages better convergence to an optimal solution. New parents are then selected from this generation, to create the new generation, and the process iterates until reaching an optimal solution, or a pre-defined minimum fitness function.

Others have suggested improvements to this initial framework. While Holland uses both crossover and mutation in each iteration, Davis [32] suggests that it is better to use a higher rate of crossover at the beginning, and higher rates of mutation as the algorithm progresses, to allow variation when the population gets stale. De Jong (cited by Reeves and Rowe, p.45) recommends that rather than potentially discarding or diluting a good solution, *elitism* should be enforced. This involves preserving the best individual created so far by placing it in the next generation in tact, and creating fewer children as a result.

Another problem is that populations can very quickly become similar, especially when the initial population is too small or invariant. Davis suggests using a no duplicates policy, that will not allow individuals into the next generation that are identical to current individuals. This idea is similar to that of the tabu search method mentioned above, and is therefore subject to the same problems with increased memory and necessary computation.

Genetic algorithms have already been demonstrated by Toyama *et al* [9] as a viable method for this type of problem, and they appear to work very effectively. There are however, a number of problems that can prevent convergence on a good solution. The key elements to getting solutions in a

reasonable amount of time seem to be the choice of a suitable fitness function, and a sufficiently diverse population. If this method is employed in this project, careful initial consideration will have to be given into these areas if good results are to be obtained.

Other Local Search Methods

Blum and Roli mention a number of more recent and less well-known algorithms based on a local search. These include the Greedy Randomized Adaptive Search Procedure (GRASP), which is a combination of a constructive method, and then a local search. Initial solutions are constructed bit-by-bit, by selecting elements from an ordered list, with the best potential elements near the top. This solution is then typically improved by using a basic local iterative search. While a simple technique, this method probably is not viable for this project, as the aim is to avoid constructive techniques where possible.

Also mentioned is the Iterated Local Search (ILS), a method which performs a local search in order to reach a local maximum, and then perturbs the solution and performs another local search. The perturbation should be just enough to move the algorithm away from the local maximum, but not too much, or it will turn into a random-restart method. Here is the difficulty in applying this method - it could be problematic to find the right perturbation method.

Finally, the Ant Colony Optimization (ACO) method is mentioned. Originally suggested by Dorigo *et al* [34], it involves simulating the behaviour of ants in finding the best paths in terms of network routing. Generally, it is more applicable to dynamically changing graphs, and therefore is not really a viable method in this project.

2.3.2 Comparing Search techniques

There are many examples of comparisons between searching techniques available in the literature. Talbi and Muntean [25], mentioned previously, compare the performance of Genetic Algorithms, Simulated Annealing, and hill climbing. They conclude that hill climbing solutions are the fastest, but of the poorest quality. SA outperforms GAs in terms of speed, but this is purely in their application to a small set of problems.

Hart [35] gives a more theoretical comparison of evolutionary algorithms (including GAs) and SA. He concludes that evolutionary algorithms will generally outperform Simulated Annealing, but only after a sufficiently large number of iterations. It is pointed out that the number of iterations required for this may be unreasonably large - therefore Simulated Annealing may

perform better for most problems. He also points out that hybrid strategies are known to outperform the individual techniques in many cases.

Blum and Roli conclude in their study that "population-based methods (i.e. GAs, ACO) are better in identifying promising areas in the search space, whereas trajectory methods (SA, TS) are better in exploring promising areas in the search space." They recommend the use of a hybrid method to combine the strengths of both. A hybrid method could potentially be implemented in this project, though it would require additional research to determine how to effectively combine different methods.

Reeves and Rowe [33] also mentions hybridization of genetic algorithms, and suggests that "the effectiveness of a GA can nearly always be enhanced by hybridization with other heuristics". They suggest including either a neighbourhood search or simulated annealing in the GA cycle. The idea is to create some offspring using crossover and mutation as usual, but then apply the NS or SA to the offspring to improve them before doing another cycle. The key parameters for success in this case would be how many of the offspring undergo improvement, whether to aim for a local optimum with the NS, and maintaining suitable population diversity.

2.4 Conclusions

After looking into the possible methods for feature extraction that are available, there are several that are promising for use in this project. Particularly, thresholding, edge detection and the Hough transform (or a variation) for extracting the document strips from a scanned image. For the more general feature extraction needed when reconstructing the document, the simpler techniques such as colour matching appear to be more appropriate, although more advanced methods like the Hough transform or snakes may be experimented with.

The key elements of choosing the search algorithm, are selecting between a bottom-up or top-down approach. The best approach for this project seems to be a top-down approach, as it increases the probability of finding a better global solution. Also important is choosing a method that includes a balance of what Blum and Roli define as *diversification* and *intensification*. Diversification is the action of performing a good exploration of the search space, and intensification is the "exploitation of the accumulated search experience".

While GAs are good for diversification, and SA and TS are proficient at intensification, there is no single method which performs well for both. This would imply that a combination or hybrid of two techniques would be the best strategy. Further investigation and experimentation will be required to

decide if this is the most appropriate strategy, considering both speed and accuracy.

Chapter 3

Design Specification

3.1 Software Process Model

As previously mentioned, this project was spilt into two main areas — image processing, and searching. It was concluded early on that there were three main stages in the development of the software: strip extraction, fitness evaluation, and searching for solutions. These stages were naturally incremental, as before any evaluation could be done, strips had to be extracted, and searching could not be done effectively without first having a fitness evaluation function.

Consequently, software development was spilt into these three stages:

- Strip Extraction: Extracting scanned strips from the background, and passing into a data structure containing each individual piece
- Fitness Evaluation: Comparing configurations of strips to determine which combination is most likely to be correct
- Document Reconstruction: Searching through possible solutions and evaluating them, in order to output a suggested final solution

This incremental process allowed for testing at each stage before proceeding to the next. As a result, there are results from these intermediate testing stages also included in this report.

3.2 User Interaction

The purpose of the project was to automate the process of recombining shredded documents, so user interaction with the system once the algorithm

is running should ideally be kept to a minimum. It is conceded though, that some user interference may be necessary, particularly in the strip extraction stages.

In terms of setting the parameters of the system, the user should be able to choose some elements for themselves where necessary — there will be some areas where a particular method will not always be the most effective, especially when dealing with different types of document. The system should therefore make allowances for this by having default settings, but permit a user to specify settings should they choose to.

3.3 Programming language

Matlab was chosen as the language to write the software in. While languages such as Java and C are generally faster, Matlab provides many useful native functions for dealing with both images and matrices — which will be the main focus of the project. It was decided that, where possible, the native functions should be used, as they will be at least as efficient as anything written for this project and probably more so. This also leaves more time to develop the parts of the project which are the real focus.

3.4 Functional Requirements

Requirements are differentiated by the use of *must* for essential requirements, and *should* for desirable, but non-essential requirements. The basic program will include all essential requirements, and subsequent improvements or changes may include some or all of the non-essential requirements.

3.4.1 Input requirements

The main input to the system will be a scanned image or images of the shredded strips of some sort of document. Ideally, the strips should not have to be ordered in any particular way prior to being scanned — they will not have to be the correct way up, or placed in specific locations on the page.

- The program must be able to accept scanned .bmp images.
- The program should accept any size of scanned image (within memory constraints), in as many file formats as necessary (i.e. .jpeg, .png, etc.).
- The program should accept as many types of document as possible (i.e. just text, text and pictures, just pictures, etc.).

- The program will not accept cross-shredded documents, only those in long strips.

3.4.2 Processing Requirements

Extraction

- The process of extracting images should be as automated as possible (i.e. the user should not have to specify the locations of the strips on the image).
- The program must extract all the relevant strips from a scanned image, excluding all other irrelevant pixels.
- The program must extract the strips into an initial data structure to work from.
- All extracted strips of the same image must be of identical size and shape.

Fitness Evaluation

- The program must process the combined image using feature extraction or Optical Character Recognition (OCR) in order to evaluate how close it is to the original document.
- The program must output the evaluation as a measure between 0 and 1.

Document Reconstruction

- The program must use some manner of heuristic technique in an attempt to improve the initial estimate by searching through the possible combinations of strips, and use some sort of evaluation to determine if an improvement has been made.
- The program should be able to process double sided documents, provided the user specifies which strips match up with each other.
- The program should be able to process multiple documents, and present the results as separate images.

3.4.3 Output Requirements

- The final output of the program must be an image, and must be readable if the original document included text.
- The final output image should be suitable for use in an OCR program if it includes text.

3.5 Non-Functional Requirements

- The program must take a reasonable amount of time to run.
- The code must be as thoroughly tested as possible to remove errors.
- The code should be clear and well-commented, and modular if necessary.

3.6 Evaluation of Output

While clearly the aim of the project is to produce a program that will output the completely correct solution — strips in the right order, and correct orientation, and potentially separated into the right documents — it was conceded that some sub-optimal solutions may also be acceptable. The degree of noise and consequent potential for difficulty in recombining the documents meant there was the possibility of producing imperfect solutions.

Therefore, a program that produces the correct solution with one strip out of place or upside down could be considered successful, as the main purpose of this investigation was to prevent the task having to be done by hand. If the majority of the work could be done automatically, only leaving perhaps two or three piece swaps to be done by a user, the program could be considered partially successful.

This will be considered when evaluating the results at the end of the project.

3.7 Project Plan

It was initially concluded that there were 4 main milestones to complete in the project – the Literature Survey, the Initial Program Development, Modifications and Additions to that, and the Project Write-up. Therefore, a Gantt chart was created (see Appendix B) to take these into account, and time was allocated accordingly:

- Literature Survey – allocated 6 weeks, with a 2-week buffer until the deadline. This was split into a review of Image Processing literature, and Heuristic Searching literature. The write-up of the conclusions was be concurrent to the reading, so there should have been no extra time at the end needed for this, although it was allowed for. While the review was going on, it also seemed reasonable to begin preliminary testing of different techniques. This helped in the conclusions of the review, and in the understanding of the techniques being read about.

- Initial Program Development – allocated 11 weeks, with a 2-week buffer. Split into a short design stage, moving onto the Image Processing part of the program, then adding the search, and then the main testing process. It was quite likely that the image processing and searching stages would merge or overlap each other as the program was developed, but the allocated time remained constant in any case. Testing was allocated 4 weeks, but this was during the exam period, which was expected to impede progress temporarily.

- Modifications and Additions – allocated 4 weeks, with a 2-week buffer. Once the basic version of the program was complete, this period was used to both improve the program, and to add some of the possible features mentioned in the requirements. These were chosen according to time remaining, and importance.

- Project Write-up – allocated 6 weeks, with a 3-week buffer until the deadline. While notes on the project were completed in the course of the year, the official write-up only began once the program was completed.

Chapter 4

Shredded Strip Extraction

4.1 Introduction

Before any work could be done on reassembling shredded strips into a finished document, the strips (having been scanned in), needed to be extracted from the scanned image, and processed into some data structure. This would contain each individual piece separately, which would then make it easy to compare strips in the evaluation stages.

The initial assumptions were that strips would be rectangular, and roughly the same height and width. This seemed reasonable, as documents put through a paper shredder would have strips the same width since the cutters are the same width apart, and assuming the document was rectangular to begin with, all of the strips would be the same height. Any strips which were very irregularly shaped were generally the edges of a document, and in this case were ignored.

As it turned out, many strips — especially those from documents made of thinner paper — have some element of curvature to them, so were not perfectly rectangular. We took account of this, and several possible methods of extracting the strips were developed.

4.2 Initial Extraction

4.2.1 Scanning in a Document

The first stage in extraction was to take a shredded document, and scan in all the strips. It was decided that strips would be stuck onto a piece of card, or paper, and scanned in that way — this would prevent any movement of the strips while being scanned in. Images were scanned at 200 dpi (dots per inch)

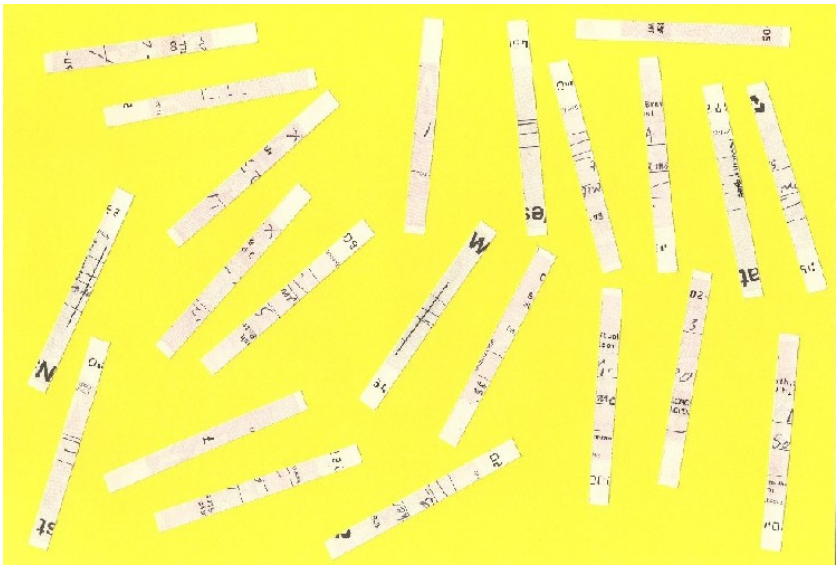


Figure 4.1: A scanned image of some strips, showing the distinctive yellow background card

which, while not being as high a resolution as might have been desirable, was adequate, and suitable in terms of Matlab memory constraints.

A distinctively yellow card was chosen for the background (Figure 4.1), as this would make distinguishing between the strips and the background more straightforward. The assumption was that none of the strips would contain so much yellow that they would blend in with the background. Later on when some strips did contain large amounts of yellow, other distinctive colours (such as blue and pink) were also used as background colours.

4.2.2 Thresholding

Once a document has been scanned in, it is thresholded, to separate the actual strips from the background. It was decided that an accurate method of choosing a threshold value could be done by taking an average of a selection of the background colour, and considering the variation in the range of colours. This was done by creating an eigenmodel based on colour, using the first and last rows of pixels in the scanned image (based on the assumption that these will be made up of just the background colour). Due to the nature of the scanning process, images were cropped so that they only included the relevant scanned image, and no irrelevant background.

Eigenmodels are statistical descriptions of data which can be useful for many applications, including Point Distribution Models, used for locating shapes in images [38]. The model consists of the origin of the data points, the best

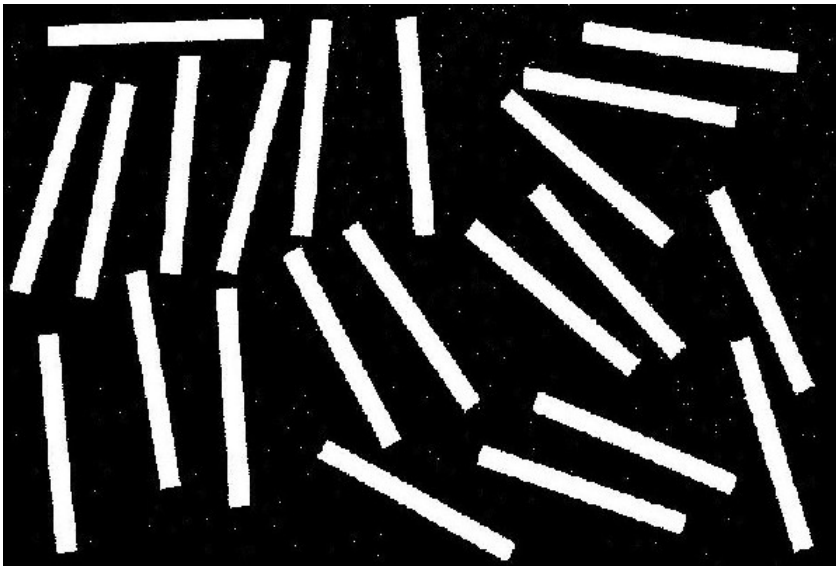


Figure 4.2: The image in Figure 4.1 after being thresholded. Several individual pixels remain, scattered among the strips.

linear fit for the data passing through the origin, and the standard deviations of the points projected onto the corresponding vector (also known as the mean μ, the eigenvectors U, and the eigenvalues of the correlation matrix of the mean-shifted data V).

Once the eigenmodel is created, the Mahalanobis distance is calculated. This is defined as:

$$d(x, y) = \sqrt{(x - \mu)^T S^{-1} (x - \mu)} \tag{4.1}$$

where S is the covariance matrix of the distribution (also defined as $U^T V U$), and x is a vector from the distribution. Having calculated the Mahalanobis distance for each pixel, all pixels in the image are scaled to a value between 0 and 1 by dividing by the maximum distance. Then, any pixels below a certain value are considered background, and the rest are considered to be strips.

The output is a binary image consisting of mainly strips, and sometimes a few individual pixels. The choice of thresholding value does still vary depending on the input image — some trial and error can be required in order to find a value that gives an acceptable result, with not too many stray pixels. Also, while some images may initially appear to not contain any of the background colour, once they were scanned in it became obvious that small areas of the background did appear.

As a result, thresholding does still require some user input to determine a value which gives suitably clear individual strips. It seems difficult to resolve this problem though, as there can be no general threshold when scanned images can vary greatly. Possibly a program could analyse the results to determine how many separate objects are in the thresholded image — too many or too few might indicate an incorrect threshold. Another method might involve not scaling the pixels to values between 0 and 1, and using a value such as standard deviations to determine the outlying pixels.

However, this seemed too trivial an extension to the program to seriously consider, as there were many other more important areas of the project that warranted further investigation. In future though, this might be a consideration. Often, a threshold value of 1/100 was sufficiently accurate, and this was generally used as an initial value.

4.3 Extracting Rectangular Strips

The initial assumption was that strips would be rectangular, so the method proposed was to fit a rectangle around each strip — all strips could then have the same dimensions and the right shape. This would be a more robust method than trying to fit individual lines to the edges of each strip, which might have given distorted final shapes.

Before the rectangles were fitted, some basic morphological operations were performed on the thresholded image. First, any small holes that might exist within the strips were filled. Also, any individual pixels were 'cleaned' away, as they would not be part of the strips, and therefore unnecessary to the strip extraction.

Each connected object (group of pixels adjacent to each other, implying a solid object) was numbered in a new array, and any objects below a certain pixel count were disregarded. This was on the assumption that strips would be of a fairly large size, so any objects below that size were unimportant. The output of these operations was a matrix of numbers corresponding to each object in the image, the number of significant objects, and their respective numbers in the matrix. Each of these objects could then have a rectangle fitted to them in turn.

Several methods for fitting the rectangles were implemented, all based on the assumption that any rectangle would have 5 main parameters: width, height, a centre point (with 2D image space coordinates) and a rotation value (Figure 4.3).

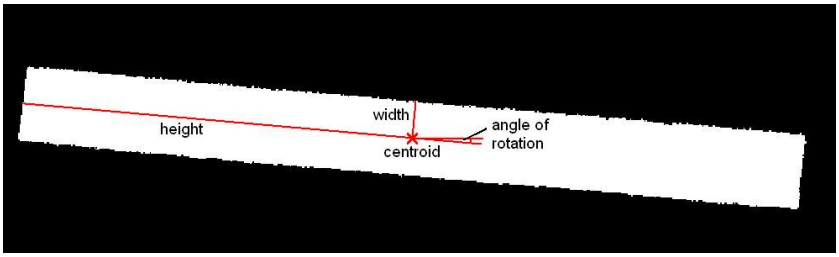


Figure 4.3: The possible parameters of a rectangle: width and height are measured from the centroid, and rotation is taken as the size of the angle rotating the strip from the horizontal

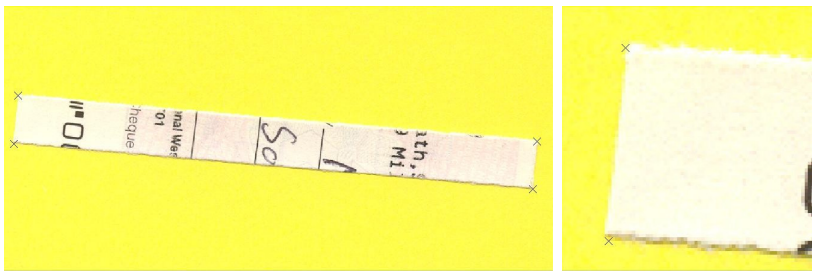
Method 1 — Points Lying on a Line

This was first method that was implemented. The centre point of the rectangle was assumed to be the centroid of all the pixels in a strip. The width and height were estimated using prior knowledge of the size of the strip — it was assumed these values could be calculated automatically later. Given these initial estimates, the program then iterated through all possible values for the rotation value, and all values within a certain range of the width and height, to find the most accurate set of values.

Calculation of the accuracy of the fitting was done by performing an edge detection on the strip to find the true edges, then calculating the corners of the rectangle being fitted. These corners were used to calculate the equation of the line joining them — the edge of the rectangle. For each pixel on the edge of the strip, it was then estimated if it lay on one of the lines of the fitted rectangle, using the equation of each line to determine if the line passed through the point (or very close to it). The rectangle with the maximum number of pixels lying on the edges would be chosen as the best fit.

This method gave fairly accurate results (see Figure 4.4), although it was decided more accurate methods were probably available. It also took a long time to iterate through all possible combinations of variables, especially rotation, as a strip could be rotated between 0 and $\pi/2$. This was later reduced to rotation between either $3\pi/4$ and $\pi/4$, or $\pi/4$ and $3\pi/4$, depending which of the differences between the maximum and minimum values of the 2D coordinates was the largest — this would indicate if a strip was more vertical or horizontal.

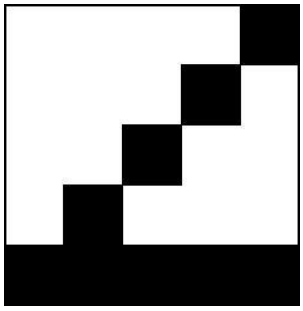
An alternative to iterating through all values was considered, in the form of choosing the initial parameters, then choosing a random combination of other possible parameters. The fitting of the new rectangle would be evaluated, and if it was an improvement on the original, it would become the new selection. This random search could go on for a set number of iterations, or until a specified fitness for the fitting had been reached.



(a) Fit on one strip with crosses marking the chosen corner points of the rectangle.

(b) Zoom in on corner points - the fit is not exactly on the real corners

Figure 4.4: Rectangle fitting using the line method



(a) Image in pixel form

0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
1	1	1	1	1

(b) The same image as a matrix

2.83	2.24	1.41	1.00	0
2.24	1.41	1.00	0	1.00
1.41	1.00	0	1.00	1.41
1.00	0	1.00	1.00	1.00
0	0	0	0	0

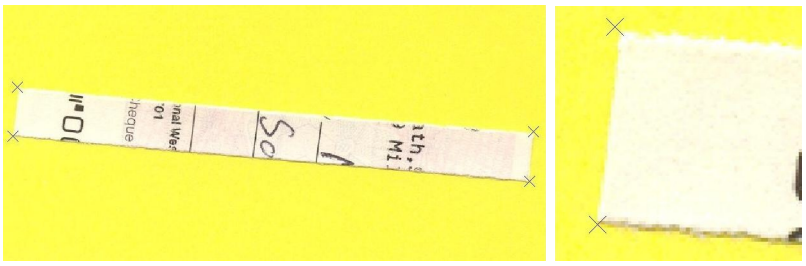
(c) The image after `bwdist` has acted upon it: All black pixel values are set to 0, and all other locations include the distance to the closest black pixel

Figure 4.5: Example of the `bwdist` function

This approach did work well on some occasions, but on others only produced mediocre results. Understandably, it was not as reliable as a systematic search, but it was quicker.

Method 2 — Distance Calculation

Another method similar to the first was also implemented, involving one of Matlab's native functions, `bwdist`. This calculates the distance of each zero valued pixel to the closest nonzero pixel, and outputs as a matrix. So, if an edge detection is done on a strip, and `bwdist` is performed, the output is a matrix with zeros where the edges of the strip are, and relevant distances everywhere else (Figure 4.5)



(a) Fit on one strip with crosses marking the chosen corner points of the rectangle.

(b) Zoom in on corner points - the fit is not exactly on the real corners, but close

Figure 4.6: Rectangle fitting using the eigenmodel method

The rectangle estimate is made in a similar manner to the previous method, and the edges are drawn in a new image. This results in a matrix of ones where the edges are, and zeros elsewhere. These two matrix images are then multiplied using `immultiply` (which multiplies the respective pixels, rather than doing a traditional matrix calculation). This gives a resultant matrix with good numerical indication of how accurate the fit is — if the fit is exact, the result should be all zeros. This matrix can be averaged, and the lower the result, the better the fit.

This method is also fairly accurate, however, it can take much longer to run than the first method, because of all the matrix computation. It might be possible to reduce this time by only operating on the necessary sections of the image. The majority of the image will be zeros, as the current strip will only be a small part, so disregarding the irrelevant parts would probably considerably speed up the process.

Method 3 — Eigenmodel

It was considered that the inherent properties of an eigenmodel would be well suited to determining the correct parameters of the strips. The mean value would be the centre point of the rectangle, and the eigenvectors would give the principle directions — which could be used to estimate the rotation, and help to calculate the width and height.

The centre point and eigenvectors can be used to rotate and translate the strip so it lies horizontally. The width and height can then be estimated using the maximum and minimum 2D coordinate values again. This solves the problem of finding an initial estimate for width and height of the strip, and gives an accurate estimate of rotation and the centre point. This method gave accurate results without too much error (Figure 4.6), while not requiring a large degree of computation.

4.3.1 Extracting the Rectangle

Once the rectangle had been accurately fitted, the coordinates of the corners were then used to create a structure containing all of the extracted strips. This was done using a plane-to-plane homography (a 3x3 homogeneous matrix). The matrix is calculated by using the coordinates of each pixel on the original image, and the coordinates of the same points on the new image. The correct rotation and translation can then be performed to move the strip to its new position.

The homography matrix, H, is based on perspective projection, using homogeneous coordinates. Any image can be transformed using rotation and translation, from one set of coordinates to another. Linear equations are set up by writing H in vector form as

$$\mathbf{h} = [h_{11}\ h_{12}\ h_{13}\ h_{21}\ h_{22}\ h_{23}\ h_{31}\ h_{32}\ h_{33}]^T$$

and H is found by solving the linear equation $Ah = 0$, with

$$\mathbf{A} = \begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 & -X_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1Y_1 & -y_1Y_1 & -Y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -y_2X_2 & -X_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2Y_2 & -y_2Y_2 & -Y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nX_n & -y_nX_n & -X_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -x_nY_n & -y_nY_n & -Y_n \end{pmatrix}$$

where x_n and y_n are the old coordinates, and X_n and Y_n are the new coordinates. In this case, n = 4, as there are 4 sets of x and y coordinates (the 4 corner points of the image).

This equation can be solved using Singular Value Decomposition (SVD), a method of solving singular linear equations. The vector h that minimises $|Ah|$, subject to $|h| = 1$, is given by the eigenvector of least eigenvalue of A^TA. This is obtained from the SVD of A. SVD calculates three matrices, generally denoted U, Σ, and V^T, where U is an nxn unitary matrix, Σ is a nx9 diagonal matrix, and V a 9x9 unitary matrix. The values of H are taken from the final column of V, which corresponds to the least value in Σ.

Once the homography matrix has been obtained, each pixel in the new image is calculated by multiplying the new coordinates by H^{-1}, (which gives the corresponding coordinates for that pixel in the old image) and using bilinear interpolation to calculate the final value (Figure 4.7).

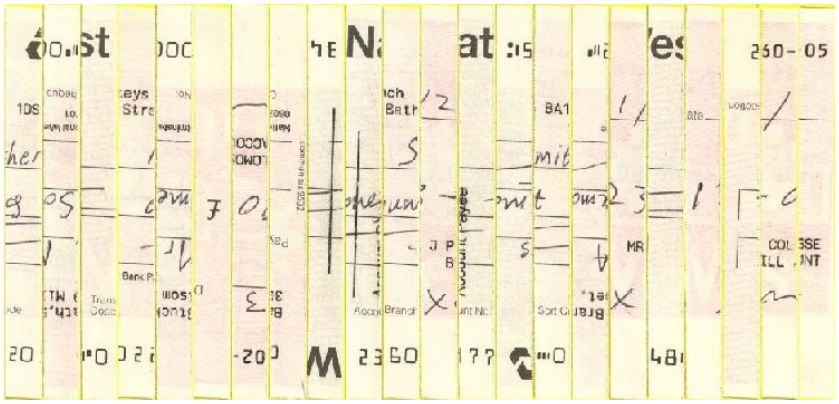


Figure 4.7: All strips from Figure 4.1, having been thresholded and extracted using rectangle fitting. Some yellow background remains, but most strips line up correctly.

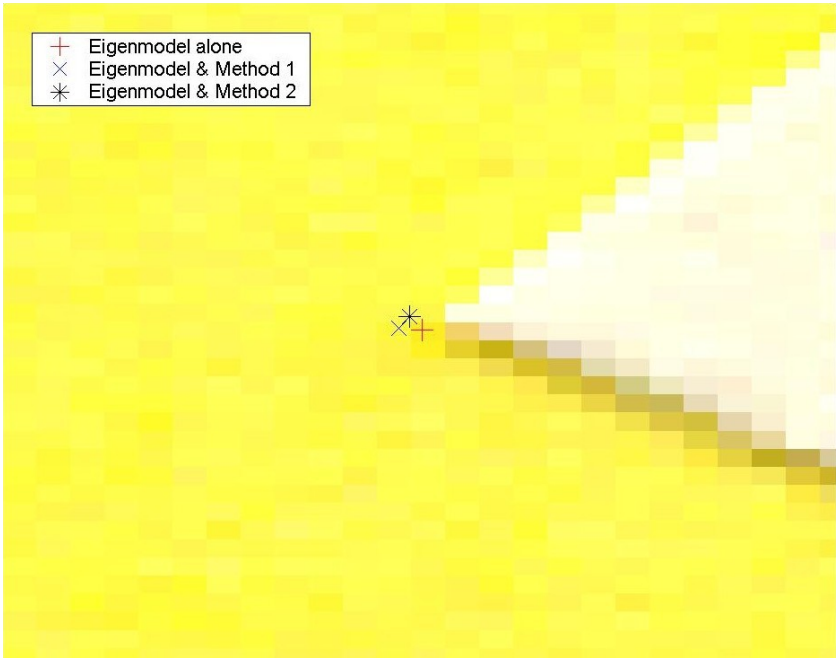


Figure 4.8: Comparing the different extraction methods: While there is some difference in the results with the various methods it is minimal, and there is little noticeable change in the final extraction

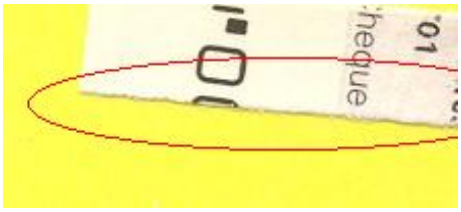


Figure 4.9: Shadow Effect: Often when scanning strips, particularly made from thicker paper, a shadow was visible on the scanned image. This made extraction more difficult, as the shadow was seen as part of the strip when thresholding

4.4 Comparison of the methods

As the eigenmodel method is fast and accurate, it was chosen as the main method for fitting a rectangle to the strips. Some experimentation was also done on refining the initial estimates by combining the eigenmodel method with the previous methods. The eigenmodel method would give an accurate estimate of all parameters, and either method 1 or method 2 could then be used to improve on it. However, the need for identical height and width of all strips meant that these parameters were already set by the eigenmodel, and consequently, using method 1 and method 2 made very little noticeable difference to the results (Figure 4.8).

Generally, fitting the rectangles to the strips was fairly accurate, although often shadows produced by scanning the strips (Figure 4.9) meant that the initial thresholding gave wider shapes than were correct, and this led to more background colour being included than was desirable. Goldberg *et al* [6] mentions a similar problem, and suggests colour photocopying of the images before scanning to reduce this. However, this may reduce the image quality and lead to a loss of colour information.

It was decided that it would be very difficult to successfully extract a strip without including any of the background colour, but this could be compensated for in the fitness evaluation stages anyway, so would not be of too much concern.

4.5 Extracting Curved Strips

After shredding several images in order to obtain a database of images to test upon, it was noted that occasionally the strips would not be entirely rectangular, but have a noticeable curve to them. This was especially evident when scanning in documents made from thinner paper. Consequently, the previous method used to extract the strips was too inaccurate, and a different

Figure 4.10: Some scanned strips with curved edges

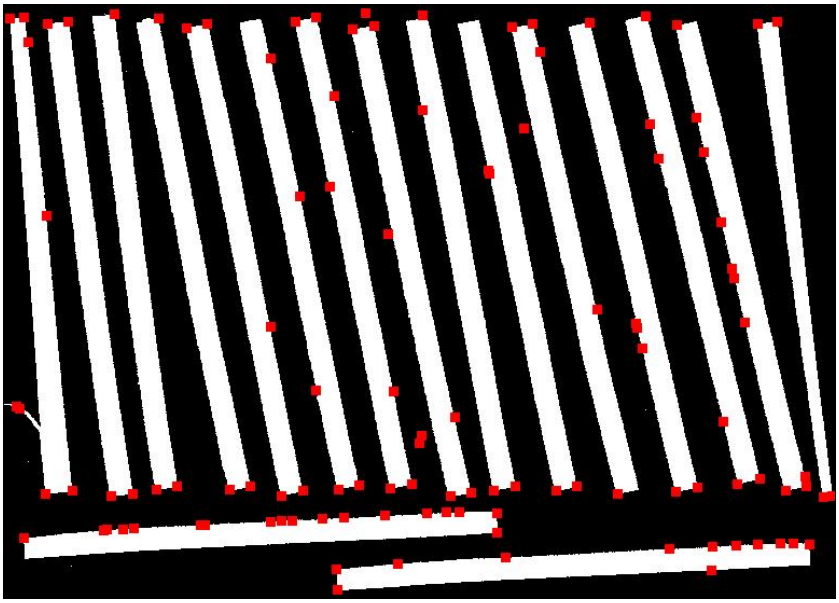


Figure 4.11: Example of Harris Corner Detector when applied to a thresholded image. While many corner points are correctly identified, there are multiple points picked out that are irrelevant.

method had to be developed. It was assumed that only two of the edges would be curved, with similar curvature (as the strips would still be the same width), and the other two edges would be straight.

The scanned strips were thresholded as before, and the corners were extracted. Several methods were considered for this, as corner detection is itself an issue of research, with some of the most popular methods including the Harris corner detector (Section 2.2.1), and the SUSAN detector. While a version of the Harris detector was tested, the results were not entirely reliable (Figure 4.11).

It was concluded that a simpler method could be used which took advantage of properties of the scanned images that were known — namely that the corner points would be approximately the maximum and minimum 2D coordinate pixel values (disregarding noise). These were used as initial es-

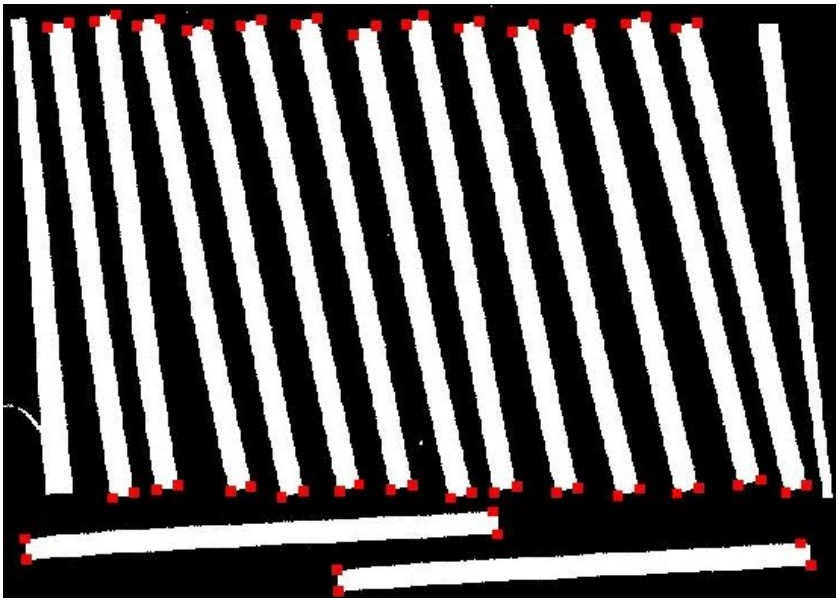


Figure 4.12: Simple corner detection on the same thresholded image as Figure 4.11: Irregularly shaped strips are ignored, and other corner points are identified as maximum and minimum strip coordinate values.

timations of the corners, which were then improved upon. Clearly, this method would be impractical to use generally, but gave acceptable results in this instance (Figure 4.12).

The corner points were used to construct a chain code along the long side of the strip, which gave the coordinates of each point on the curved edge. A curve was then fitted to these points as closely as possible, using the native function in Matlab (`polyfit`). This gave the curve in the form of an equation of a polynomial curve, with the values for a, b and c:

$$f(x) = ax^2 + bx + c \tag{4.2}$$

Having fitted the curve, the input and output values (the coordinates of the points in the curve) were calculated so that each pixel would be at an equal distance along the curve itself — this would avoid distortions in the final image. Each pixel in the new strip was calculated by stepping along the curve in equal increments — for each pixel, its distance along the curve would be found using the polynomial curve, and therefore its coordinates were calculated. This gave the basic position of the pixel in the original image, and bilinear interpolation was used to calculate the correct pixel colour more accurately. Then, the starting point of the curve was shifted to the next pixel along the width of the strip, and the process was repeated.

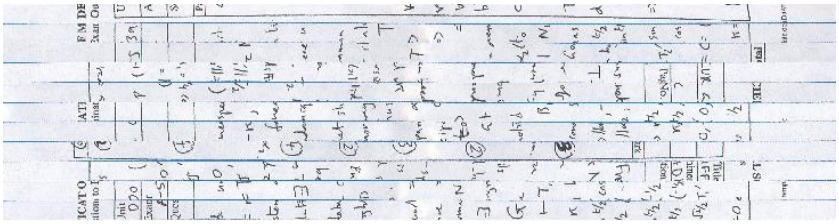
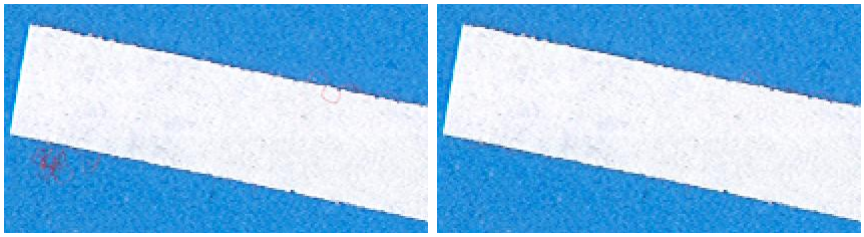


Figure 4.13: All strips from Figure 4.10, having been thresholded and extracted using curve fitting. Some blue background does remain, but the fit is generally better than rectangle fitting.



(a) A scanned image with a hair trapped under a strip

(b) The same image after manipulating the image to hide the hair

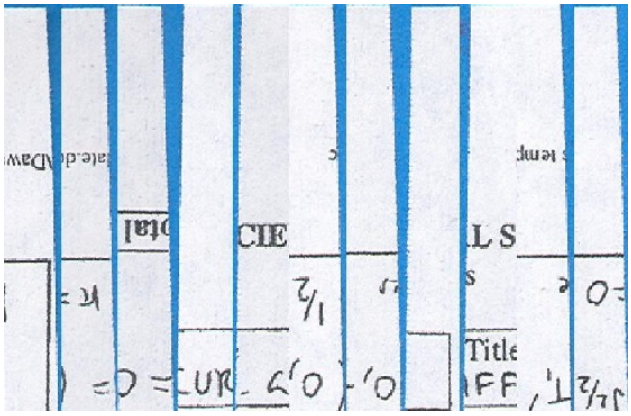
Figure 4.14: Image manipulation to make fitting more straightforward

This process was based on the assumption that the curve was not a true parabola, and therefore that a single input might give multiple outputs which would confuse the program. However, observation showed that if a strip did show curvature, it tended to curve purely in one direction. Also, strips were scanned in at a slight slant (to fit them on an A4 page), so the problem did not occur. These assumptions did mean that the method was less robust than it could be though, and might fail when used in more general circumstances.

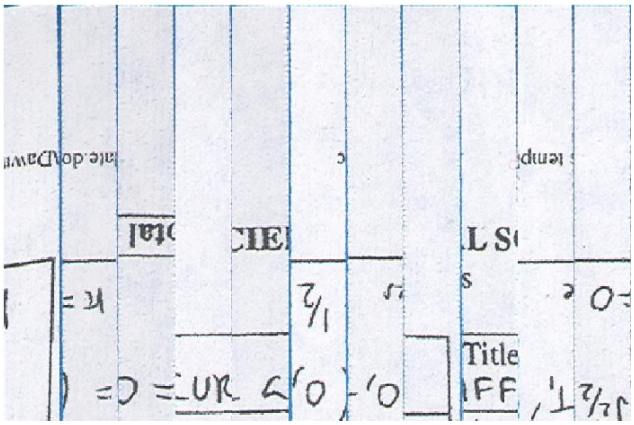
Results were generally accurate (Figure 4.13), although occasionally the polynomial curve fitting was slightly inaccurate. Calculating the correct width was also often difficult and sometimes edges of strips would be slightly cut off. Occasionally, ill defined edges of strips, or edges that were very ragged (this occurred particularly when shredding cardboard) would prove problematic for the curve fitting method — the chain coding would get stuck in a loop, or the fit of the curve to the points would be inaccurate. In some cases, this could be solved by manipulating the original images to remove offending objects (for example, a hair trapped under one of the strips while being scanned in, as in Figure 4.14).

4.6 General Comparisons

Overall, the rectangle fitting works better with strips made from thicker paper or card, as they tend to keep straight edges. Also, often shredded card strips have jagged edges, which are difficult for the curve fitting method to deal with. Both methods give acceptable results, but generally the curve fitting includes less of the background, and the strips line up better (Figure 4.15).



(a) Strips from Figure 4.10, extracted using rectangle fitting



(b) Strips from Figure 4.10, extracted using curve fitting

Figure 4.15: Comparison of strip extractions: When extracting curved strips, the curve fitting works more accurately, including minimal blue background, and lining strips up well

4.7 Conclusions

While both extraction methods have their weaknesses, generally they perform sufficiently well to justify focusing on other areas of the project. Areas for improvement would include making the curve fitting more robust — this would probably involve better corner detection, as the current methods are not resistant to noise.

In terms of extensions to the methods, something to be considered might be dealing with more irregularly shaped strips. Often paper can become jammed in a paper shredder, leading to strips not entirely separated. Also, there are often slightly smaller, strangely shaped strips at the edges of shredded documents. It would be useful to be able to include these strips when recombining documents, as they could provide useful extra information.

In fact, it might be beneficial to take irregular shapes into account when recombining documents. If one strip has a part missing from one side, and another strip has a similar sized extra part to it, it seems likely that the two fit together. Using the extra information gained by taking size and shape into account could be a useful tool — this relates back to the idea of recombining jigsaw puzzles (Section 2.1.2).

Chapter 5

Fitness Evaluation

5.1 Introduction

Having successfully extracted all strips, the next stage was to develop a metric to compare the strips, and evaluate the likelihood of strips being paired adjacent to each other. Methods used previously in other studies were considered, as well as new techniques. The main methods considered were generally comparing individual pairs of strips, however it was noted that methods of comparing complete solutions might also be helpful.

5.2 Proposed Methods

Initial Evaluation

It was decided that evaluation of synthetic data, that of pictures cut into strips artificially by the computer, would be useful initially when designing an evaluation metric. While strip extraction was generally accurate, often some background colour still remained around the edges, and the strips did not match up perfectly. Using synthetic data would mean all strips would line up exactly, and there would be no noise.

Clearly, noise and background colour would have to be accounted for when comparing strips in the real program, but comparing evaluation techniques would be easier to do without this additional noise. If the data was synthetic, when comparing evaluation techniques, it would be clearer to see which genuinely outperformed the others based on performance of the algorithm alone.

Having used synthetic data, performance comparisons could then be done with real data, and useful conclusions drawn about the overall performance of the techniques.

5.2.1 Pixel Comparison

Both Toyama and Kosiba use a basic method of fitness evaluation, matching pairs of pixels along the edges of each strip. This is based on the principle that when comparing strips that are adjacent to each other, the colour values along the edges should be very similar. Therefore, the initial fitness evaluation we used simply compared each pixel along the edge of a strip with the corresponding pixel on the other edge.

Comparing the colours was done using the basic method of comparing the difference in colour values of each pixel — those with similar colours would gain a low score, and very different colours a high one. These comparisons were averaged over the entire strip, and a final fitness evaluation obtained by subtracting this from 1, giving a fitness range from 0 to 1.

Several distance measures were contemplated for computing the difference between colour values. Mancas-Thillou and Gosselin [39] in their study of colour text extraction from background, mention several possibilities: Euclidean, Manhattan, Chebychev, Minkowski, Mahalanobis, Canberra, Cosine, etc. They also note the difference between clustering distances and similarity measures — distances can range from 0 to ∞, while similarity measures range from 0 to 1.

Any metric suitable for the type of direct comparison being done were considered:

Euclidean:

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}. \tag{5.1}$$

Manhattan (or City Block):

$$\sum_{i=1}^{n}|x_i - y_i|. \tag{5.2}$$

Chebychev:

$$\max_i|x_i - y_i|. \tag{5.3}$$

Canberra:

$$\sum_{i=1}^{n}\left(\frac{|x_i - y_i|}{x_i + y_i}\right). \tag{5.4}$$

Cosine:

$$\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}}. \tag{5.5}$$

A variation of Euclidean distance was also considered, where the $\sqrt{}$ is left out, for computational efficiency. This referred to as Non square-rooted Euclidean (NSR Euclidean).

Storing the Evaluations

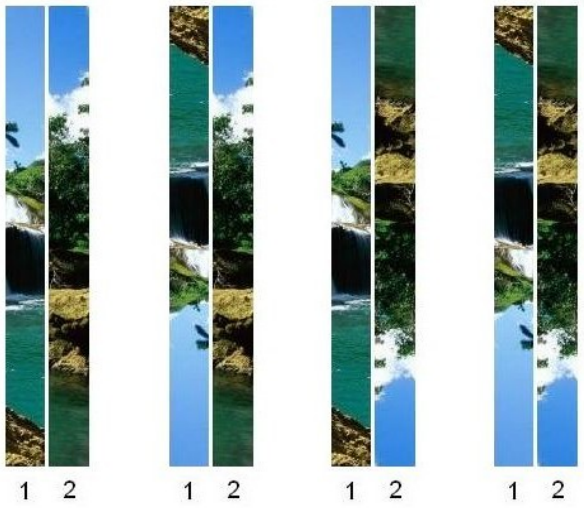
One advantage of only comparing pairs of strips meant that the total number of evaluations would only be $2n(n-1)$ for an image with n strips — taking rotation into account. This meant that all comparisons could be calculated and cached into a fitness table, which could then be used by the searching algorithm as a lookup table. This would save time while running the GA, as comparisons would not have to be calculated as it ran, but just found in the comparison table.

In fact, the fitness table contained $4n(n-1)$ values — each comparison was stored twice. This was due to the possible rotations of the strips. Each pair of strips could be compared in 4 different ways: both correct way up, one strip rotated (either the first or the second), or both rotated (Figure 5.1). However, these comparisons are equivalent to the same comparisons for the strips in reverse order (Figure 5.2). To make the lookup more straightforward when searching, it is easier to store these comparisons twice than to work out their equivalent ordering before being able to look them up.

Evaluating the Metrics

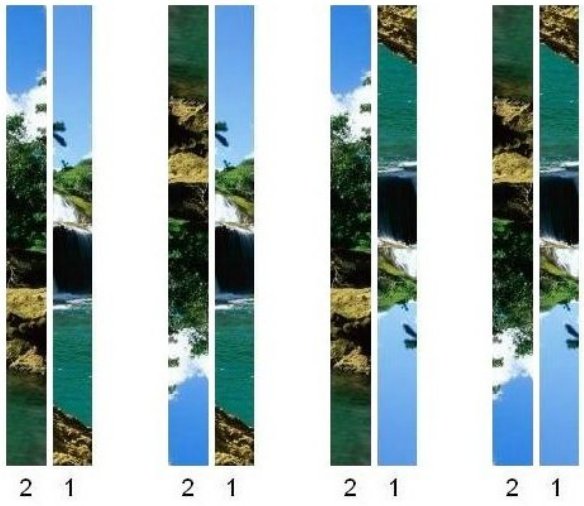
The fitness table could also be used to determine the effectiveness of the evaluation. Having calculated all fitnesses, these could be sorted from lowest to highest. Ideally, the correct comparisons would be at the top of this sorted table (the smallest distances). Taking the mean and standard deviation of the correct fitnesses, gave a measure of the accuracy of the fit.

Evaluations which had a mean above the target value would have a higher proportion of false positives (incorrect comparisons with better evaluations than true comparisons), and false negatives (correct comparisons with worse evaluations than incorrect comparisons). Standard deviation gives a good indication of how widely varying the accuracy of comparison is. For testing, an image was split into 25 strips, and a fitness comparison was done for each possible pair of strips. These fitnesses were then sorted — the aim would



(a) Both strips original way up (b) First strip rotated (c) Second strip rotated (d) Both strips rotated

Figure 5.1: All possible ways of comparing two different strips, when considering rotation



(a) Both strips original way up (b) First strip rotated (c) Second strip rotated (d) Both strips rotated

Figure 5.2: Identical strips to those in Figure 5.1, with the ordering reversed. It can be seen that Figure 5.1(a) is identical to Figure 5.2(d) when rotated 180 degrees. Consequently, when comparing the edges, the result would be the same for both sets of strips.

be for the evaluations of the correct pairings to be the top 24 in this sorted table, making the mean around 12.

Another method of seeing more visually how well the fitness evaluation performed was to plot it onto the image (Figure 5.3). At each edge comparison, the accuracy of the evaluation was plotted as a square ranging from black to white — black meaning it was a very good fit, white being very poor. Ideally, for the correctly ordered image, comparisons would all be black (Figure 5.3(a)), while for an incorrect image comparisons would be a range of colours (Figure 5.3(b)). If the correctly ordered image contained white squares, it would show that the fitness evaluation worked ineffectively.

Comparing the different distance measures, (Figure 5.4), we can see that typically, there is very little difference in accuracy. Using the HSV colour space, evaluation is very accurate (Figure 5.4(b)), but no more so than RGB colour space (Figure 5.4(a)). When evaluating an image with little colour variation (Figure 5.5), results are less accurate (Figure 5.6), but still within acceptable levels. Looking at the plot of the effectiveness for each image, the edge comparisons are all close to black (Figure 5.7).

After some consideration, it was concluded that HSV offered no obvious advantages over RGB, and given the extra computation required to convert images to HSV it was decided that RGB would be the better choice. Likewise, none of the distance measures significantly outperformed the others, but the Chebychev distance measure was generally chosen over the others, because of the reduced computation involved. However, the option of choosing the other measures was kept in the final program.

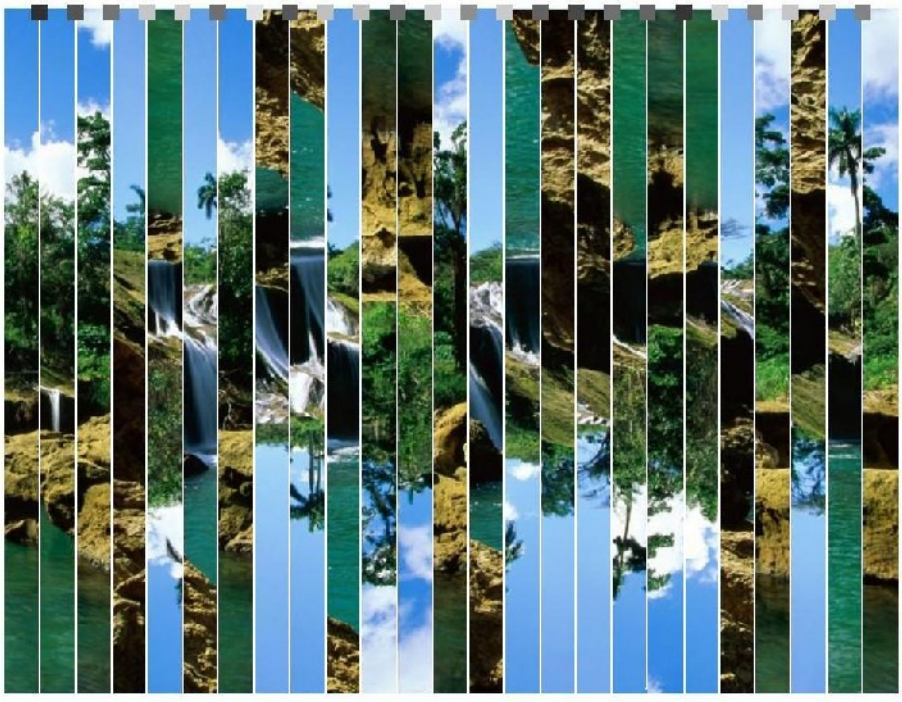
5.2.2 Scale Space Comparison

It was considered that a more efficient fitness evaluation would not only consider the pixels on the very edge of strips, but also further inside — this would be especially true for the real data, as noise and background pixels would almost certainly impede evaluation at the edge. Therefore, an extension to the evaluation was considered, that of comparing strips at different scales. If strips were compared at half their normal size, pixels at the edge would include a proportion of data from further inside the strip. By continually halving the sizes of strips and comparing them, eventually all data on each strip would be included in the evaluation (Figure 5.8).

Evaluating the Metrics

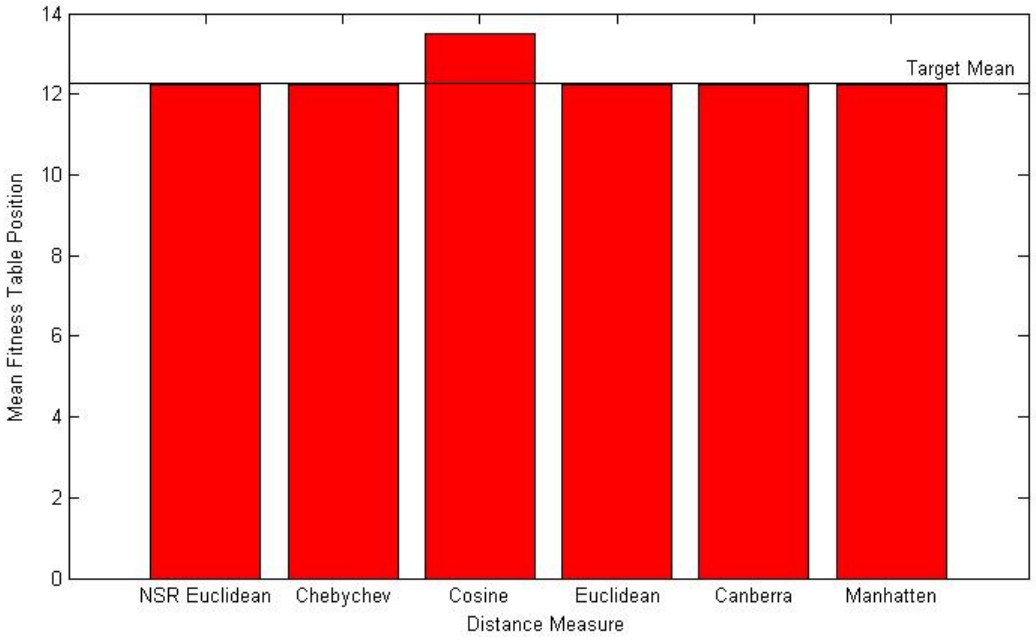
Consideration was required to determine the proportion of each scaled evaluation to include. Several different weighting ratios were considered (Figure 5.9).

(a) Correctly recombined image

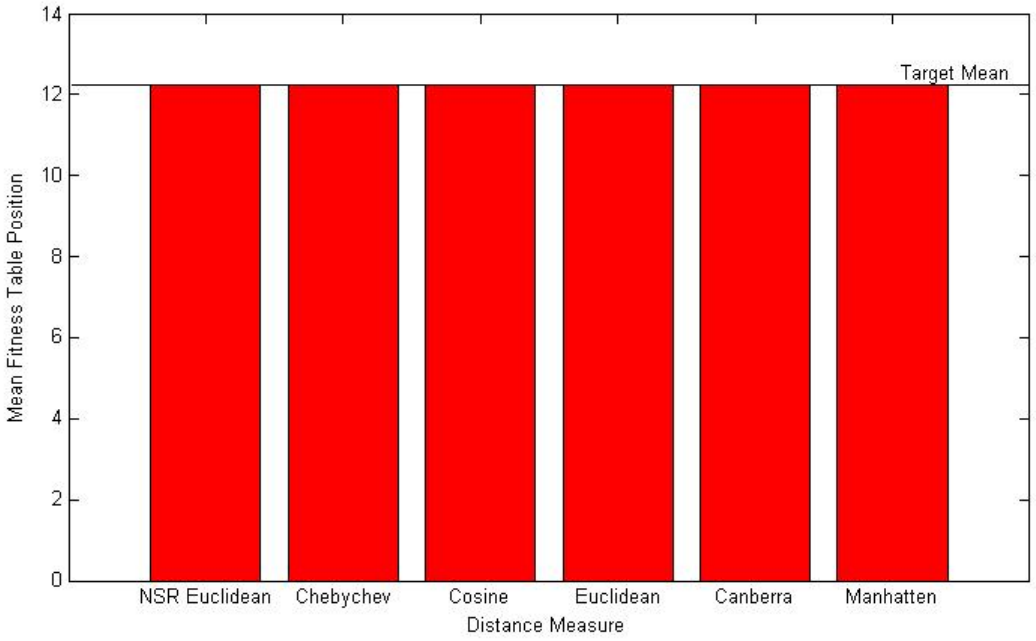


(b) Incorrectly combined image

Figure 5.3: Visual representation of the accuracy of the fitness evaluation. Black squares indicate correctly matched strips, white squares show incorrectly matched strips



(a) Comparing distance measures in RGB space



(b) Comparing distance measures in HSV space

Figure 5.4: Evaluating fitnesses for a generic image: The aim is to be as close to the mean fitness position as possible - going over the mean implies a less accurate fit. On most images, the fitness evaluation works well in both RGB and HSV space for all distance measures

Figure 5.5: Synthetic testing image: Split into 25 equal strips

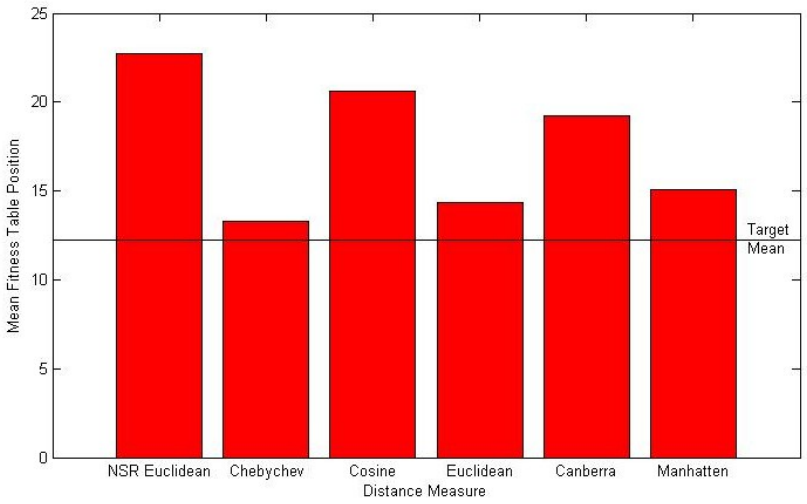


Figure 5.6: Comparing distance measures for Figure 5.5 - the image has little colour variation, so performance of the fitness evaluation is expected to be poorer. Most distance measure still stay close to the target mean value though

Figure 5.7: Accuracy of the fitness evaluation for a similarly coloured image. While the colours are similar, and it would be difficult to reconstruct this image, the fitness evaluations for each edge are generally accurate.

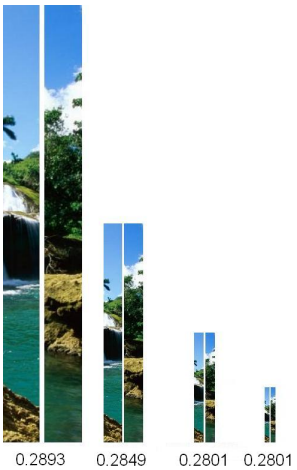
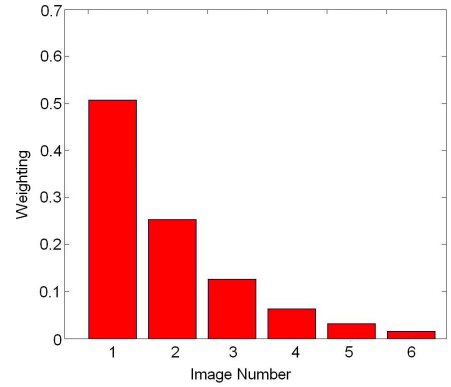
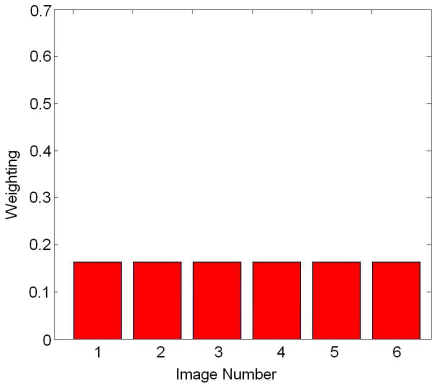


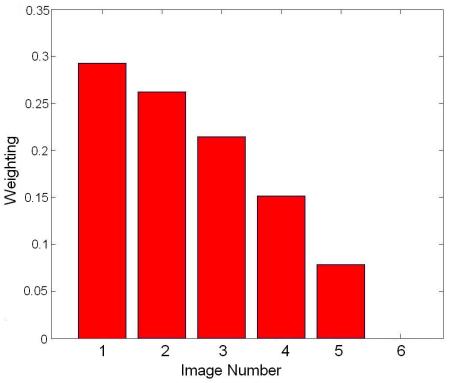
Figure 5.8: Example of scaled down piece comparison — each pair has its fitness evaluation underneath



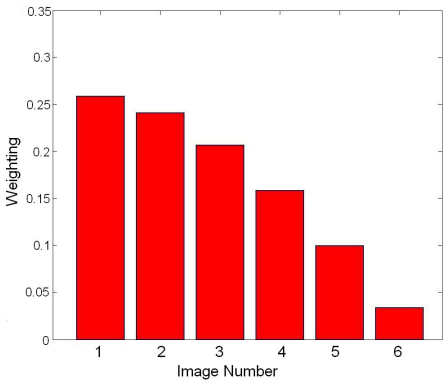
(a) 2^{n-1} (Halving the weighting each time)



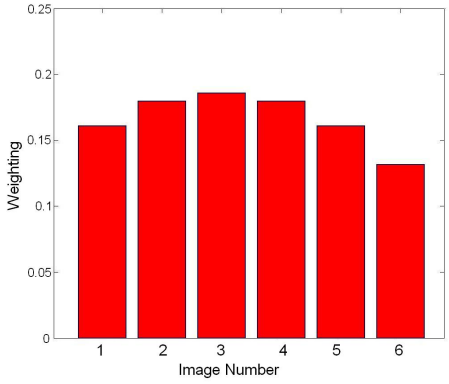
(b) 1/n (Equal Weghting)



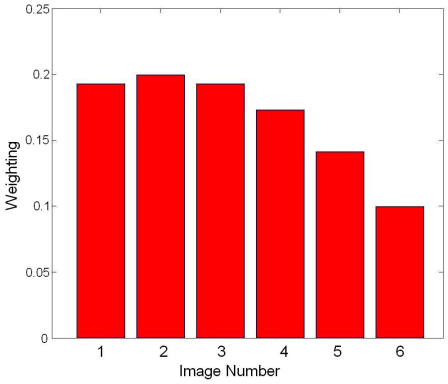
(c) $\cos(\pi/2n) + \pi/12$



(d) $\cos(\pi/2n) + \pi/24$



(e) $\cos(\pi/2n) - \pi/6$



(f) $\cos(\pi/2n) - \pi/12$

Figure 5.9: Possible weighting ratios for the scaled strips - each weighting sums to 1. For example, with the first ratio, the final fitness evaluation would be a combination of 0.5 times the full-sized fitness comparison, plus 0.25 times the half-sized fitness comparison, plus 0.125 times the quarter-sized fitness comparison, etc.

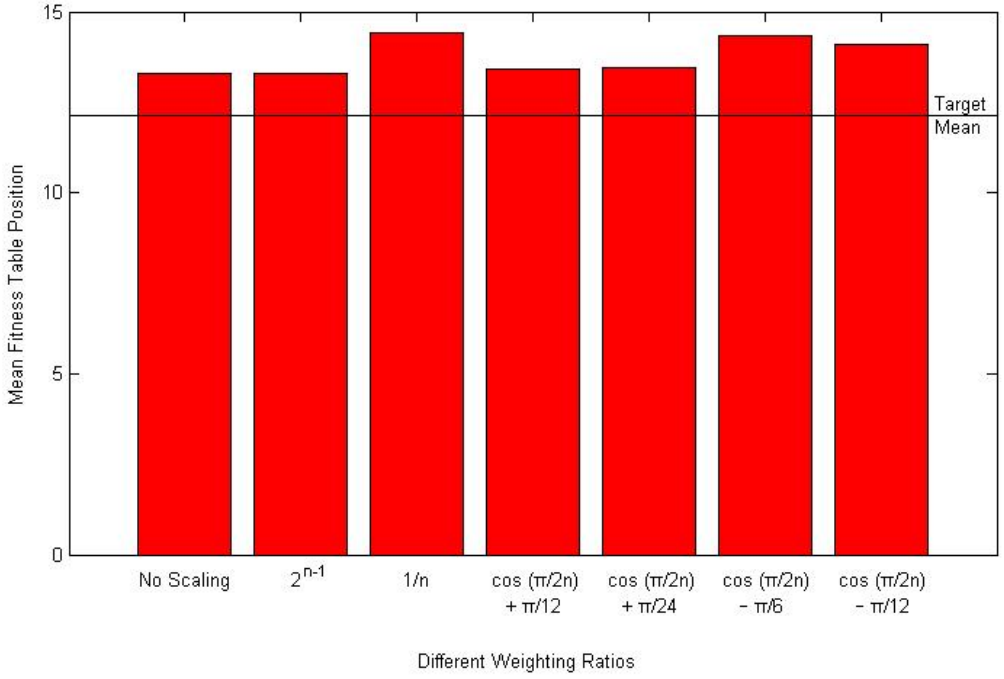


Figure 5.10: Comparing the different weighting ratios: Mean values are still close to the target for all the different ratios, and no particular weighting stands out as better than the others, or better than the performance of the fitness evaluation without scaled spaces

Similar comparisons of the results were done, using the mean fitness positions in the evaluation table (Figure 5.10).

Overall, including the scaled image comparisons provided little noticeable improvement with the simulated data, but the ability to calculate them was still included in the program, as it was considered they could be useful when comparing real data.

5.3 Reducing Noise

While these initial evaluation measures worked very well on synthetic data, performance on real data was unimpressive. It was concluded that evaluation would be more effective if background colour from the card could be identified, and removed from the fitness evaluation stage. Therefore, the eigenmodel created during the extraction stages was used to identify how similar pixels in a strip were to the background colour.

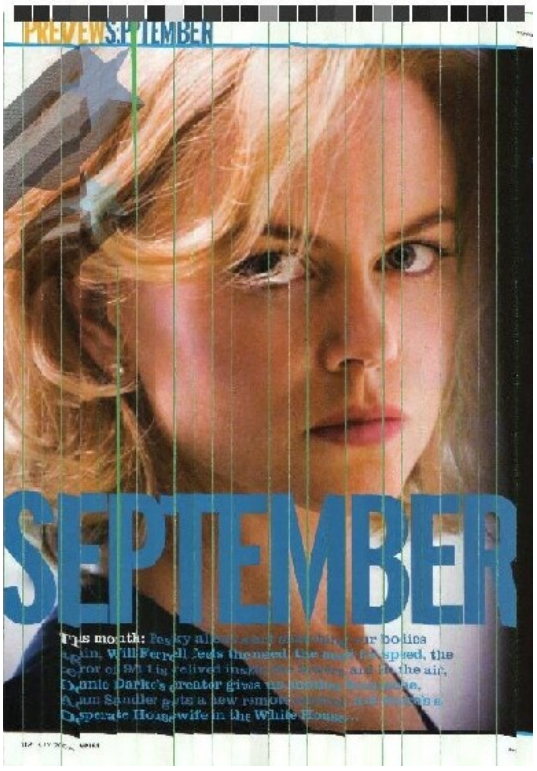


Figure 5.11: Fitness evaluations for a real image: Generally the evaluations are accurate, but a couple are poor, as shown by the white squares

Having identified these pixels, a fitness evaluation was performed which took a section of the edge of a strip and compared it with a section of the edge of another strip. Each column of pixels in one strip was compared with each column of the pixels in another strip, and comparisons were weighted according to how similar the pixels were to the background colour. The result was less emphasis on background pixels, and more on the relevant pixels.

Having achieved this improvement, experiments were performed to determine the most effective combination of pixel weighting, and image scaling. Overall, it was concluded that when comparing strips extracted using the curve fitting method, going inwards by 6 pixels gave adequate results (Figure 5.11), while comparing strips extracted using rectangle fitting required 12 pixels because of the extra background often included when using that method. Scale space comparison was done in addition to this, but generally seemed to offer little improvement in accuracy — most likely because the noise reduction step already included image information from inside the strip.

5.4 Other Possible Methods

While pixel comparisons worked reasonably well, some other methods were considered as alternatives:

5.4.1 Colour Histograms

Ukovich *et al*, in their study into this problem, advocated the use of content based image retrieval methods to try and compare strips. It was decided this should be attempted on a small scale level, by creating colour histograms for the edges of each strip. This could be expanded to include the entire strip if required, but using the edges initially was thought to give a more accurate match.

Colour histograms give an example of the colour distribution of a selection of pixels — this can include RGB distribution, or HSV, whichever is more effective. Images with similar colours will have similar histograms, and consequently, edges which match will also have similar histograms. A distance measure, as used above, can be used to decide how similar two histograms are, and consequently, how well two strips match up.

Histograms have the added advantage of being more noise invariant than the basic pixel comparison — as pixels are not matched, it does not matter if the edges of extracted strips are not completely lined up, as the colour histograms will still be similar. Also, background colour can be subtracted from the histograms if it does remain in the strips. However, this approach would probably be less effective for text or writing.

Some evaluation of the histogram method was done using the visual overlay as before (Figure 5.12). While on some images the histogram would perform well (Figure 5.12(a)), on others it would perform more poorly (Figure 5.12(b)) with more gray evaluation squares, particularly compared to the pixel comparison technique. It was concluded that more investigation would be required to determine the best circumstances to use the histogram method.

5.4.2 Edge Detection

This was considered as a possible method, especially for comparing text based documents, where using colour comparisons would most likely be less efficient. However, when some of Matlab's native edge detectors were used on various images, the performance was unimpressive on the very edges of the strips, and the edges of the strips were frequently detected. This implies that edge detection would perform poorly if used as a genuine technique (Figure 5.13).

(a) One example of histogram evaluation

(b) Another example of histogram evaluation

Figure 5.12: Histogram fitness evaluation: While the histogram method performs well on some images (a), on some others (b) the performance is inferior to the pixel comparison method (see Figure 5.3(a) for comparison).

(a) Image of part of a strip

(b) Image after Sobel Edge Detection

Figure 5.13: A typical edge detection: the edge detection does not work at the very edges of the image, and the visible edge of the strip is erroneously included

5.5 Conclusions

While the evaluation function does perform sufficiently well to allow useful comparisons of strips, it is not as effective as it could be. The problems of noise, badly extracted strips, and background colour being included mean that the comparisons are not as accurate as they should be.

However, it was concluded that the current methods were good enough to justify moving onto document reconstruction — it was certainly possible that despite less than perfect evaluations, a heuristic search might still be able to provide good solutions. After all, the nature of the problem meant that having a bad evaluation for one or two strips could be compensated for with good evaluations for all the others, as the general fitness was taken as the average of all fitnesses.

Generally, improvements to the fitness evaluation in future might make more use of colour histograms or other content based image retrieval techniques, as the results they gave were promising. Also, some sort of text recognition or language database could be used when dealing with mainly text-based documents. This went beyond the bounds of this project, but could be an area of study in the future.

It should also be worth considering an evaluation which computes the fitness of the entire document as a whole, rather than comparing pairs of strips. This project focused deliberately on top-down methods of reconstruction, as it was thought this would provide better overall results. It could be beneficial to use this for the fitness evaluation — looking at the complete picture,

rather than simply the sum of its parts. This could also be considered in the future.

Chapter 6

Document Reconstruction

6.1 Introduction

Having developed a suitable evaluation method, searching techniques were required to find the best possible reconstruction of strips. Research during the literature survey led to the conclusion that a Genetic Algorithm would be am efficient heuristic method to use, possibly developing into a hybrid method for improved performance.

Due to the similarity of this problem to the Travelling Salesman Problem, similar methods were considered, particularly in regard to the crossover functions. It was decided that tests should be carried out on both real and synthetic data, to give a good idea of the performance of the GA in general. Variables such as population and mutation could then be tailored more accurately to the real data.

6.2 Initial Genetic Algorithm

6.2.1 Solution Representation

Due to the nature of this problem, it is impossible to encode it as a binary series as Holland originally suggested. There are several suggested methods, the most obvious being as a permutation of the strips, i.e. [1 3 5 2 6 4]. However, the added complexity of the rotation of strips meant including some record of the orientation. As a result, the order was represented as a permutation in one row, and the rotation of the strip in another, with 0 representing no rotation, and 1 meaning 180 degree rotation:

$$\begin{bmatrix} 1 & 3 & 5 & 2 & 6 & 4 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

6.2.2 Initial Population

Studies have been conducted into optimal population size [40], though it is impossible to determine precisely without an idea of the particular problem. The issue is really whether to increase population, which can slow down the algorithm considerably, or have a more limited population, and risk losing diversity. Initially, it was decided that the population size would be roughly double the number of strips, though later testing concluded that a higher population size gave overall better results (see Section 6.3), with not too significant an increase in complexity.

6.2.3 Selection

Possibly the most commonly used selection strategy is the Roulette Wheel method. This involves selection of parent genomes depending on their fitness relative to the rest of population. For example, if there are two genomes and one has double the fitness value of the other, the genome with the larger fitness will have double the chance of being chosen to be a parent. In this way, the fittest members of the population have a greater chance of reproducing, and consequently lead to stronger children.

Elitism

While Holland's original concept for GAs was that every generation would only include offspring created from parents following selection, crossover and mutation, De Jong suggested this might not be the most effective method. He suggested the idea of *elitism*, and *population overlap*. Elitism involves including the best member of the population automatically in the next generation, while population overlap extends this to only creating a few new members of the population each time.

This can lead to problems with diversity, but does mean that good members of the population are not thrown away once they are created. Consequently, elitism was used when creating the populations in the program, but not population overlap, so diversity was maintained.

6.2.4 Crossover

Crossover techniques traditionally used for Genetic algorithms were unsuitable for this type of problem, as they would lead to repeating members of the population. For example, swapping half the genes of one parent with half of another:

$$p_1 = [1\ 3\ 5\ |2\ 6\ 4] \qquad p_2 = [5\ 2\ 4|\ 1\ 3\ 6]$$

$$c_1 = [1\ 3\ 5\ 1\ 3\ 6] \qquad c_2 = [5\ 2\ 4\ 2\ 6\ 4]$$

Therefore, techniques used for problems such as the Travelling Salesman problem were investigated, using Larranaga *et al* [36]. It was decided to only create one child from the combination of both parents, though this could be altered if necessary in the case of the ordered crossover technique.

Ordered Crossover

Also used by Toyama and initially proposed by Davis [37], based on building subsections of the original document, it copies a section of one parent to the child (in a random place), then fills the rest of the child with the remaining genes from the other parent in order. The rotation included in the representation does not affect the implementation of this algorithm, so all orientations are simply copied from the parent genes to the child.

$$p_1 = [1\ 3\ 5\ 2\ 6\ 4] \qquad p_2 = [5\ 2\ 4\ 1\ 3\ 6]$$

cut point:

$$p_1 = [1\ 3\ |5\ 2\ 6|\ 4] \qquad p_2 = [5\ 2\ 4\ 1\ 3\ 6]$$

$$c = [4\ |5\ 2\ 6|\ 1\ 3]$$

Edge Recombination Crossover

Developed by Whitley *et al* in Davis [32], based on the idea of keeping matching edges together. This algorithm had to be altered slightly, to allow for the rotation element included in the representation. An *edge map* is created initially which lists all the edges of both parents, in both possible orientations, when moving right along the string (see Table 6.1).

$$p_1 = \begin{bmatrix} 1 & 3 & 5 & 2 & 6 & 4 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$p_2 = \begin{bmatrix} 5 & 2 & 4 & 1 & 3 & 6 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

If we look at the entries in the table for gene 1, it has the genes to the right of the current gene (and their orientation) — this refers to the genes

Gene	Connected genes			
	0		1	
1	3	4	3	
	1	1	0	
2	6	4	5	5
	0	0	0	0
3	1	6	5	1
	1	0	1	0
4	6	1	2	
	1	1	1	
5	3		2	2
	0		0	0
6	4		2	3
	1		1	1

Table 6.1: Edge Map

to the right of the current gene in its initial orientation. So under 0, it has $[3\ 1]^T$, and under 1, it has $[3\ 0]^T$. Then, genes to the left of the current gene are considered, as they would be to the right of the current gene if it were flipped around — so under 0 is also $[4\ 1]^T$, as

$$\begin{bmatrix} 5 & 2 & 4 & 1 & 3 & 6 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

is identical to

$$\begin{bmatrix} 6 & 3 & 1 & 4 & 2 & 5 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

so the edge can be included in the edge list. As 1 is at the beginning of one of the parents, there are no other possible edges to include.

Once the edge map is created, an initial gene is chosen at random, and all occurrences of this gene are removed from the connected genes side of the table. If there are edges to choose from in the table on the row corresponding to the currently selected gene, the gene with the fewest possibilities in its own edge list is chosen as the next in the list. Any ties are broken at random. If there are no possible edges to choose from in the table, the next gene is chosen at random. This carries on until all genes have been accounted for.

6.2.5 Intelligent Crossover Methods

It was noted previously during research into heuristic methods, that using hybrid methods frequently improves the performance of searching algo-

rithms. Reeves and Rowe notes that "hybridization should be used wherever possible; a GA is not a black box, but should use any problem-specific information available" ([33] p.59), and indeed, this seems reasonable. Therefore, some more intelligent methods were developed beyond the basic crossover methods.

Intelligent Ordered Crossover

In order to implement more intelligent searching, individual fitness edge ratings for each member of the population were recorded, and passed onto the crossover functions. The intelligent ordered crossover method did a search for the best edge fit in the first parent — this was used as a starting point, and adjacent edges to this were evaluated to determine if they were below a threshold, and if they were, they became part of the section to be passed onto the child. Having chosen this section, the rest of the child was completed using the same method as before.

The performance of this method depends quite highly on the choice of threshold — too low, and only very small sections are inherited which makes the crossover rather useless, too high, and most of the parent can be passed to the child. This threshold can be difficult to judge, but generally, the mean of the fitness calculations minus one standard deviation was chosen, which worked well in most cases.

Intelligent Edge Recombination Crossover

Again, individual fitness edge ratings for each member of the population were recorded, and passed onto the crossover functions. The method created an edge map as before, but fitness ratings were also included:

Gene	Connected genes			
	0		1	
1	3	4	3	
	1	1	0	
	0.32	0.16	0.22	
2	6	4	5	5
	0	0	0	0
	0.11	0.31	0.28	0.09
⋮				

Table 6.2: Intelligent Edge Map

Edges were then chosen in a similar manner, but instead of ties being broken at random, the edge with the best fitness was chosen instead.

6.2.6 Mutation

The final stage in any Genetic algorithm, necessary for continued diversity. The simplest methods of mutation are to swap two genes in a sequence, or to switch the orientation of a single gene in a sequence (flip a piece around). Mutation rate can be difficult to optimise, but Reeves and Rowe suggests a rate of 1/n, where n is the length of a string.

Method 1:

$$\text{c} = [1\ 3\ 5\ 2\ 6\ 4] \rightarrow \text{swap 3 and 5} \rightarrow \text{c} = [1\ 5\ 3\ 2\ 6\ 4]$$

Method 2:

$$\text{c} = \begin{bmatrix} 1 & 3 & 5 & 2 & 6 & 4 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

flip 3 around:

$$\text{c} = \begin{bmatrix} 1 & 3 & 5 & 2 & 6 & 4 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

More advanced mutation methods were also considered for implementation. The first extension was to use the fitness of individual strips to choose which strip to mutate — strips with lower fitness values would be more likely to mutate. The other possibility considered was to randomly choose a strip to mutate, then possibly mutate other adjacent strips, depending on their fitness ratings — if the fitness with the strips adjacent was above a threshold, those strips would also be mutated (similar to the intelligent ordered crossover).

For example, in the previous mutation, the 3 was flipped - using the advanced mutation, the 1 and the 5 might also be flipped, depending on their fitness evaluation with the 3. Whichever edge combination gives the highest fitness will be chosen. It was thought this would help to maintain subsections of the document that might have been correctly matched. Some experimentation was done as regards mutation rate, and the different mutation methods (see Section 6.3).

6.3 Testing the Initial System

A variety of images were chosen for testing the final system (see Appendix A). Each image was extracted, all piece comparisons were pre-calculated, and used in the GA for evaluation. Images were tested with both regular and intelligent crossover methods, and the results were recorded (Table 6.3).

It can be seen that the Intelligent Edge Recombination GA outperformed the other crossover methods in general, but overall, the desired solutions were not reached, although often the final solution was close (Figure 6.1, Figure 6.5).

Image	Correct Fitness	Highest Fitness Rating Found			
		Ordered Crossover	Intelligent Ordered	Edge Recomb.	Intelligent Edge Recomb.
Tigers	0.7708	0.7801	0.7804	0.7838	0.7842
Planet	0.7490	0.8160	0.8194	0.8197	0.8197
Ape	0.8211	0.8327	0.8308	0.8327	0.8327
Cheque	0.8868	0.9074	0.9065	0.9069	0.9094
Girl	0.8237	0.8236	0.8317	0.8258	0.8324
Text	0.8501	0.8746	0.8749	0.8745	0.8778
Superhero	0.8415	0.8446	0.8383	0.8476	0.8613
Lines	0.9231	0.9168	0.9105	0.9240	0.9296
Woman	0.8462	0.8223	0.8281	0.8415	0.8498
Rainbow	0.8716	0.8656	0.8600	0.8678	0.8748

Table 6.3: Results of Testing (Real Data): The Intelligent Edge Recombination consistently performs the most effectively in finding good solutions

The main exception to this is in the case of images consisting mainly of text (Text and Cheque). This is to be expected, as the nature of the fitness evaluation is dependent on slow changes in colour across strips — which is not the case with text. While pixels will sometimes match, if there is slight misalignment, or parts of the text are cut off by the strip extraction process, matching will become difficult.

It seems that the sections of document that are correctly recombined depend strongly on the fitness evaluation (see Figure 6.8). It can be seen in Figure 6.8(a), that the poorest fitness evaluations (shown as the white/grey boxes) correspond to the edges which are not correctly matched in the output image (Figure 6.8(b)).

It is interesting to compare the results of the Text image (Figure 6.9(a)) with that of the Rainbow image (Figure 6.9(a)), which was deliberately chosen because it included similar text, but had a clearly changing background. The performance of the system on the Rainbow image was generally fairly good — the colour changes as it should across the strips. However, the text is occasionally upside down, which implies that the text data did not play a significant role in the matching.

Tests were also performed on synthetic data, to compare the results (Table 6.4). Looking at these results, it seems that the reason the system does not perform to standard is because of the fitness evaluation. When using

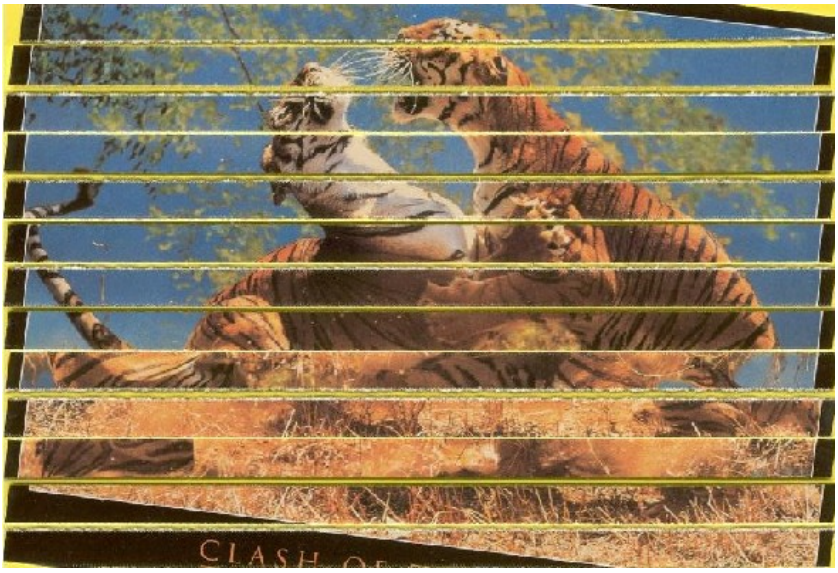


Figure 6.1: Tigers: Solution found with the highest fitness rating

Figure 6.2: Planet: Solution found with the highest fitness rating

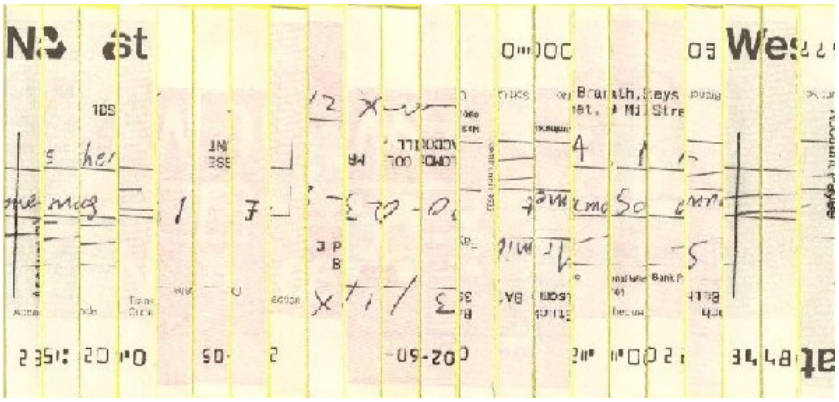


Figure 6.3: Cheque: Solution found with the highest fitness rating

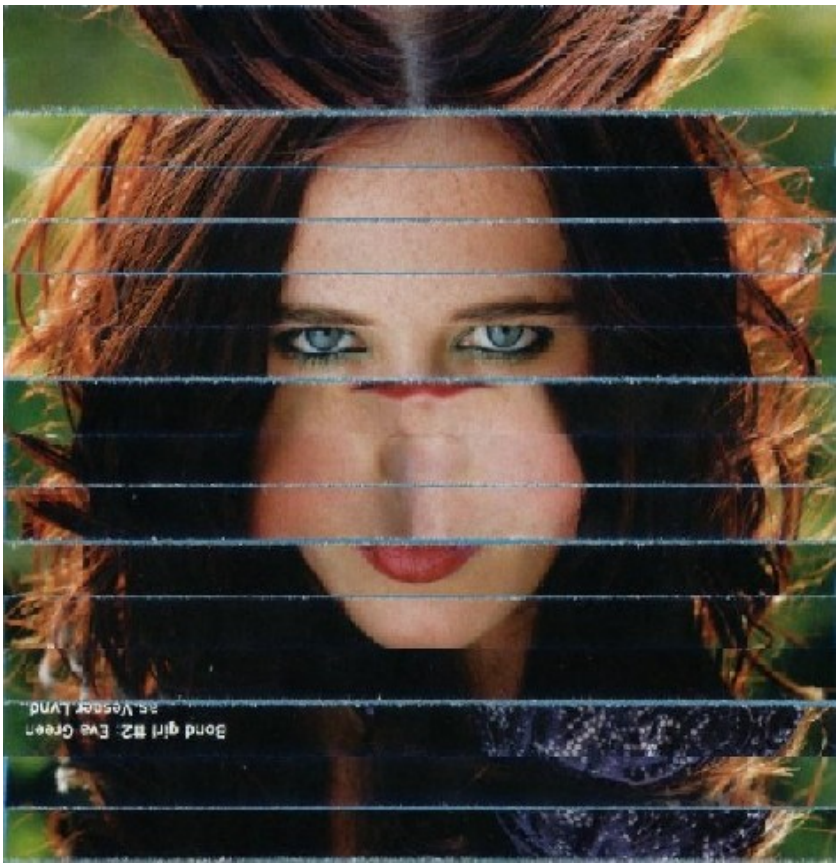


Figure 6.4: Girl: Solution found with the highest fitness rating

Figure 6.5: Lines: Solution found with the highest fitness rating

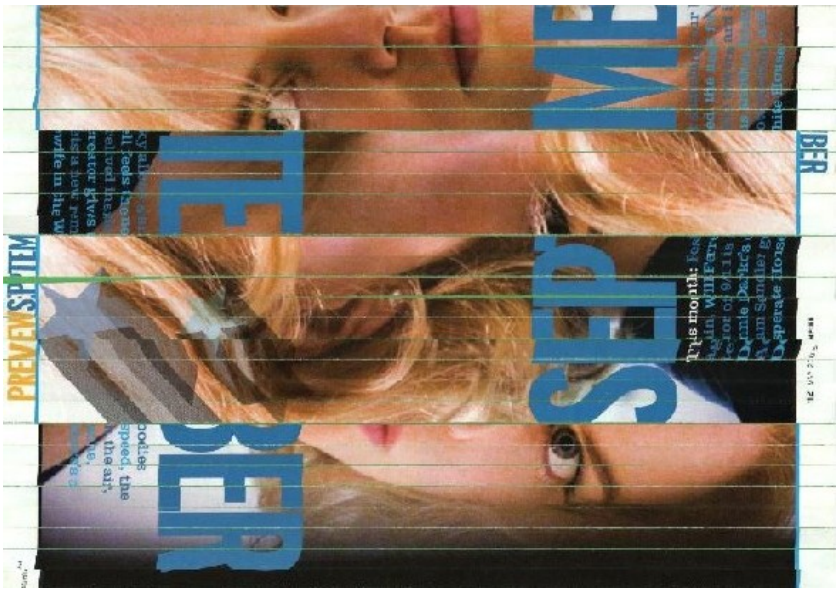


Figure 6.6: Woman: Solution found with the highest fitness rating

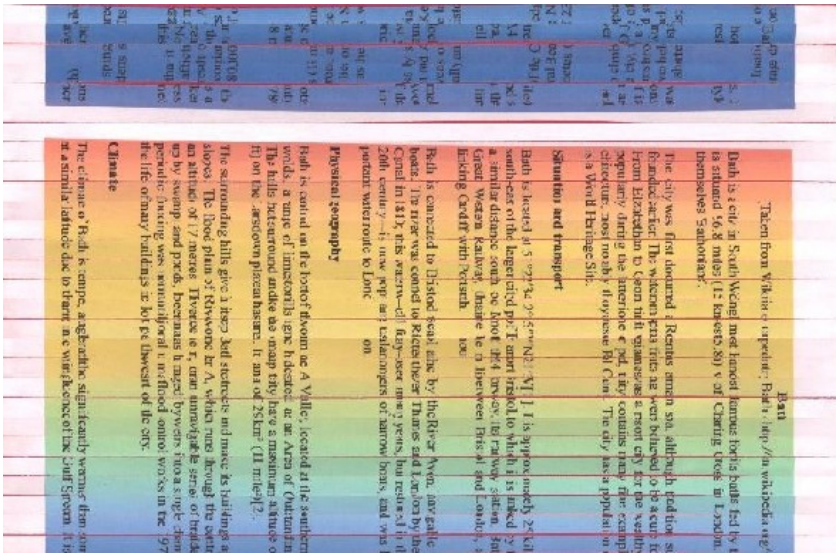
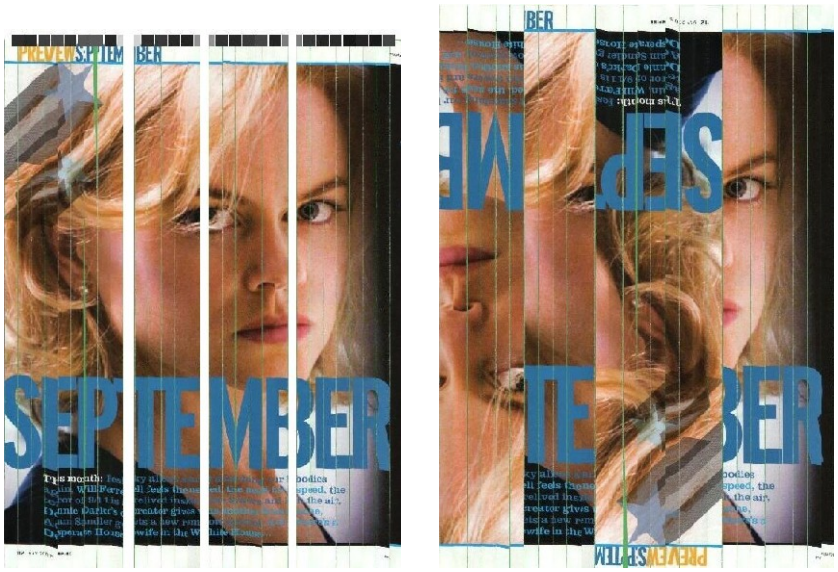


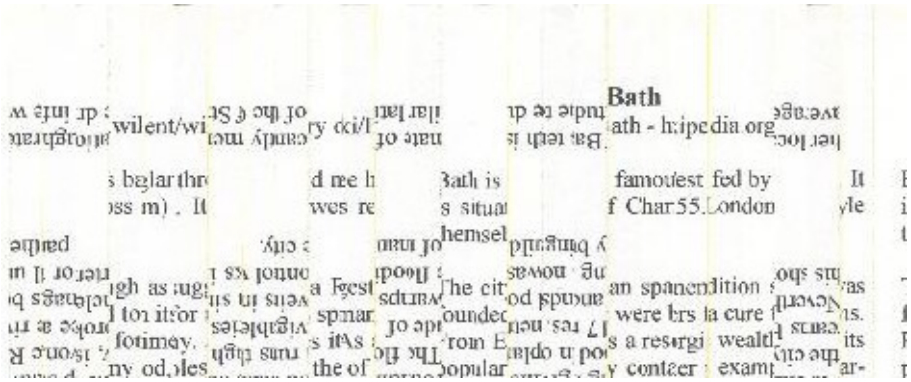
Figure 6.7: Rainbow: Solution found with the highest fitness rating



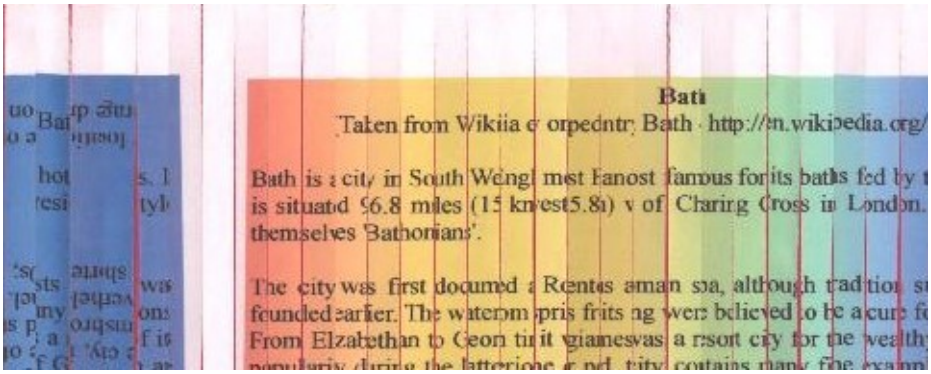
(a) Fitness evaluation for Woman image: The poorest fitnesses are separated in the image

(b) Output imge for Woman: The sections match those in part (a)

Figure 6.8: Correspondance between fitness evaluations and the final output: The poorly evaluated edges are not recombined in the final image



(a) Part of Outputted Image - Text



(b) Part of Outputted Image - Rainbow

Figure 6.9: Comparison of output from two different images: The Text image has not really been combined well, while the Rainbow image shows the correct colour change across the strips

Image(Strips)	Correct Fitness	Highest Fitness Rating Found			
		Ordered Crossover	Intelligent Ordered	Edge Recomb.	Intelligent Edge Recomb.
Waterfall(13)	0.9696	0.9532	0.9696	0.9696	0.9696
New York(25)	0.9462	0.9303	0.9412	0.9462	0.9462
Tower(20)	0.9646	0.9646	0.9422	0.9426	0.9646
Purple(15)	0.9758	0.9758	0.9758	0.9758	0.9758

Table 6.4: Results of Testing (Synthetic Data): The Intelligent Edge Recombination is consistent in finding the correct solution when dealing with synthetic images

synthetic data, and therefore very accurate fitness calculation, the GA will frequently find the correct solution. When looking at the real data, the GA cannot find the correct solution, as other solutions exist which have higher fitnesses.

More extensive testing was done using the Intelligent Edge Recombination Crossover, as it showed the best performance. A test was performed using the Woman image to compare mutation rates, over 1000 iterations with different rates of basic mutation, and the average fitness was recorded. The same test was then performed using the more advanced method of mutation (see Section 6.2.6). The results can be found in Figure 6.10.

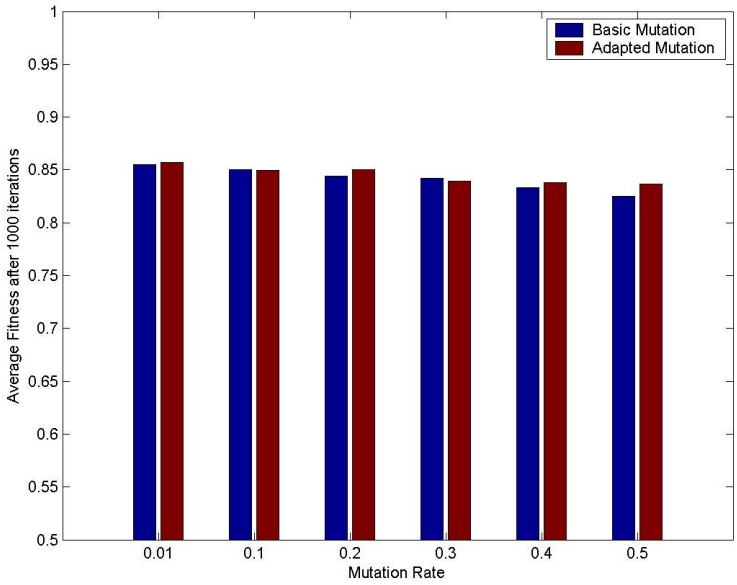
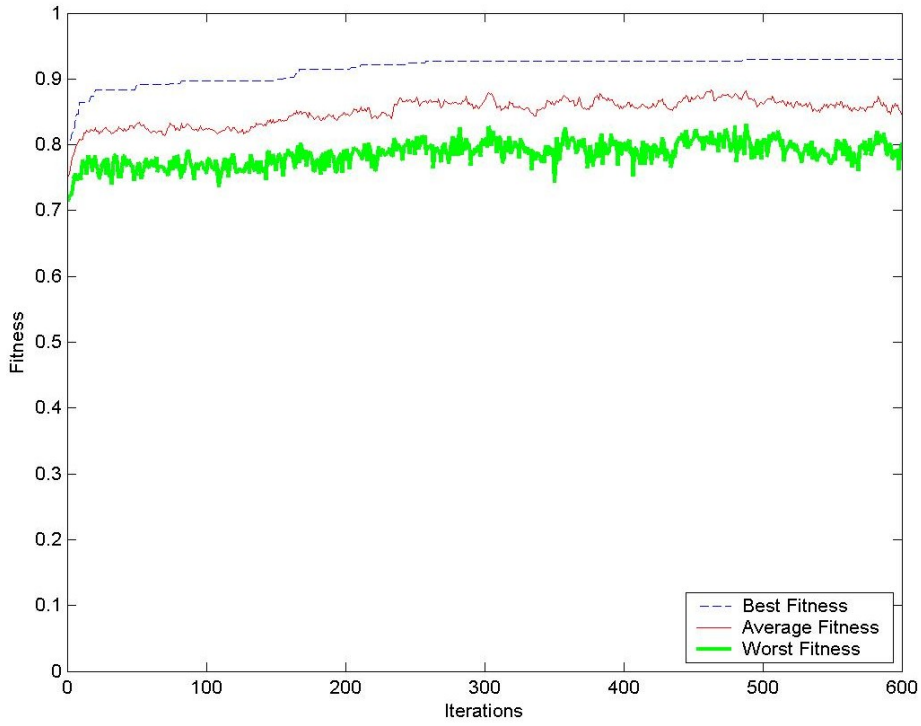


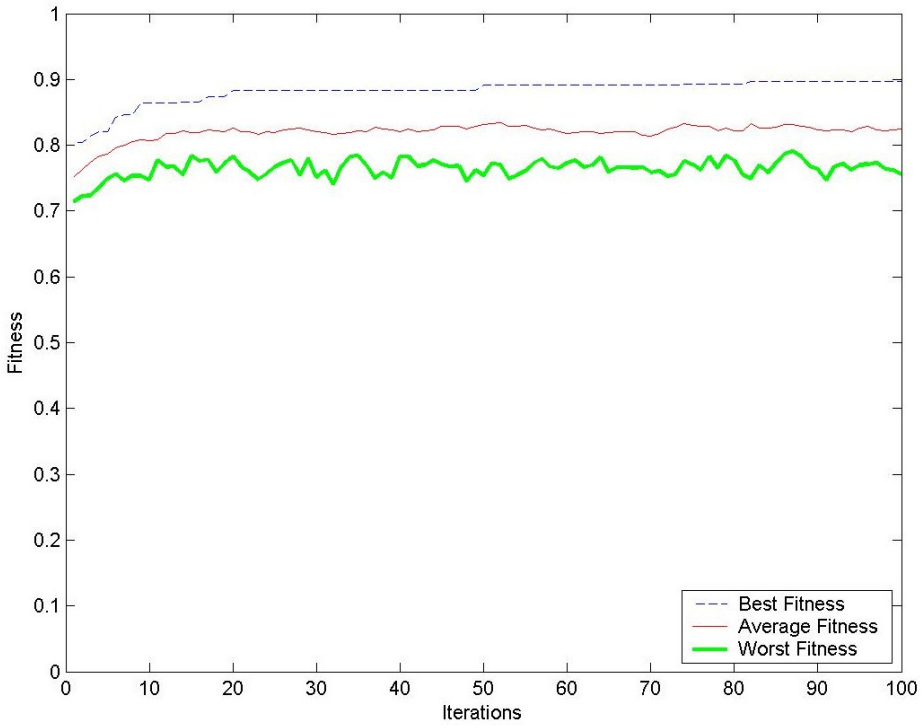
Figure 6.10: Mutation Rate Comparison: Fitness is higher with lower rates of mutation, and the more advanced methods of mutation showed slightly better results

Overall, lower mutation rates led to higher averages, and the advanced mutation outperformed the basic mutation slightly. This implies that there is sufficient diversity in the Intelligent Edge Recombination Crossover method to reduce the need for mutation — most likely this stems from the initial random choice of strip used in the method.

An example of the fitness over a typical run of the algorithm (on the Lines image) demonstrates this (Figure 6.11 and Figure 6.12). In Figure 6.11(a), the average fitness, best fitness, and worst fitness are plotted over time. All three improve rapidly at the beginning (Figure 6.11(b)), as expected, then tend to remain fairly steady for the rest of the run, rising gradually (except the best fitness, which jumps in steps, due to the elitism in the algorithm).

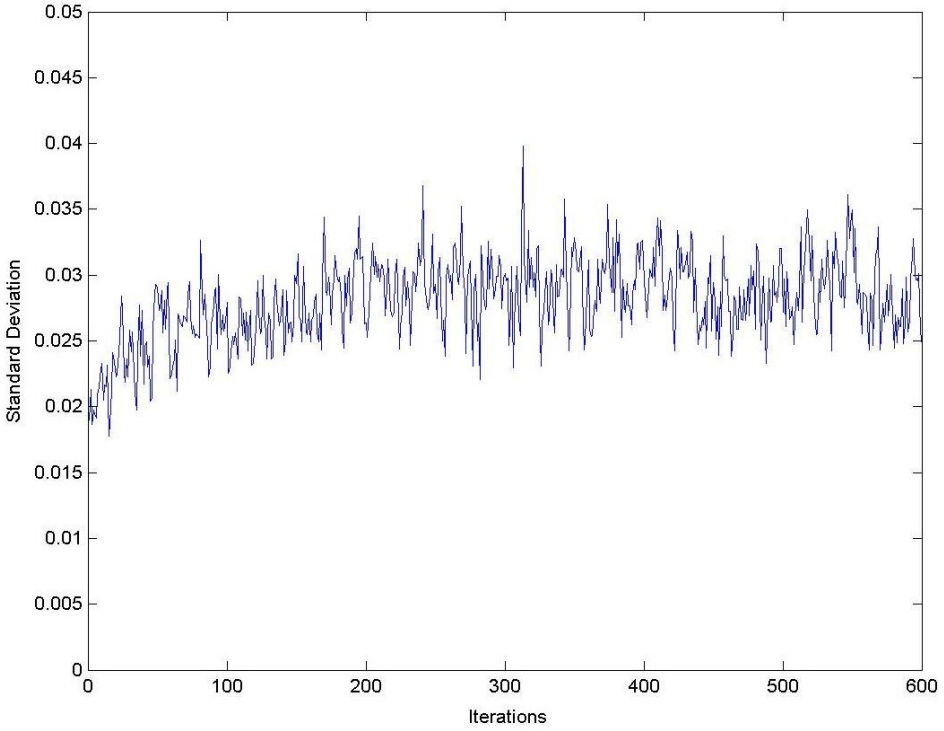


(a) Example of fitness over a run of the algorithm, on the Lines image



(b) Beginning of Figure (a): Shows the fast increase in fitness initially

Figure 6.11: Fitness of the population: Shows initially large increase, slowing over time to change very gradually



(a) Standard deviation of the fitness

(b) Lines: the image being tested

Figure 6.12: Standard deviation of the population: Variation grows over time - this is most likely because of the elitism in the algorithm. As shown in Figure 6.11(a) and Figure 6.11(b), the average and worst fitnesses are fairly static over time, while the best fitness increases - leading to the larger deviation.

In Figure 6.12(a), the standard deviation of the fitness of the population is shown to also increase initially, then remain fairly constant.

To show the improvement of the method over simpler searches, the Woman image was put through a random search, and the results plotted (Figure 6.13), versus the fitness of the Intelligent GA. The random search shows little improvement over time, while the intelligent method can converge on a good solution within several thousand iterations.

It is worth noting that while the Edge Recombination GA frequently outperforms the Ordered Crossover, it does take significantly longer to run, with the same number of iterations. Most likely this is because of the need to create an edge map for each set of parents every iteration. Therefore, if the Ordered Crossover could be made more effective, or possibly if some combination of the two crossovers could be managed (perhaps, using one crossover for half the children, and the other for the rest), a more useful crossover function could be performed.

6.3.1 Missing Strips

It was considered that while trying to reconstruct documents made from shredded strips, it is possible to have a strip missing and not realise (until after trying to reconstruct the document). Therefore, some experimentation was done on the Tigers document, with a random strip missing. Results (Figure 6.15) were encouraging — when compared with the reconstruction in Figure 6.1, the results are very similar. It seems that the particular strip that is missing does have some effect, as Figure 6.15(d) has a more inaccurate result, possibly because the missing strip was an important joining strip between other sections.

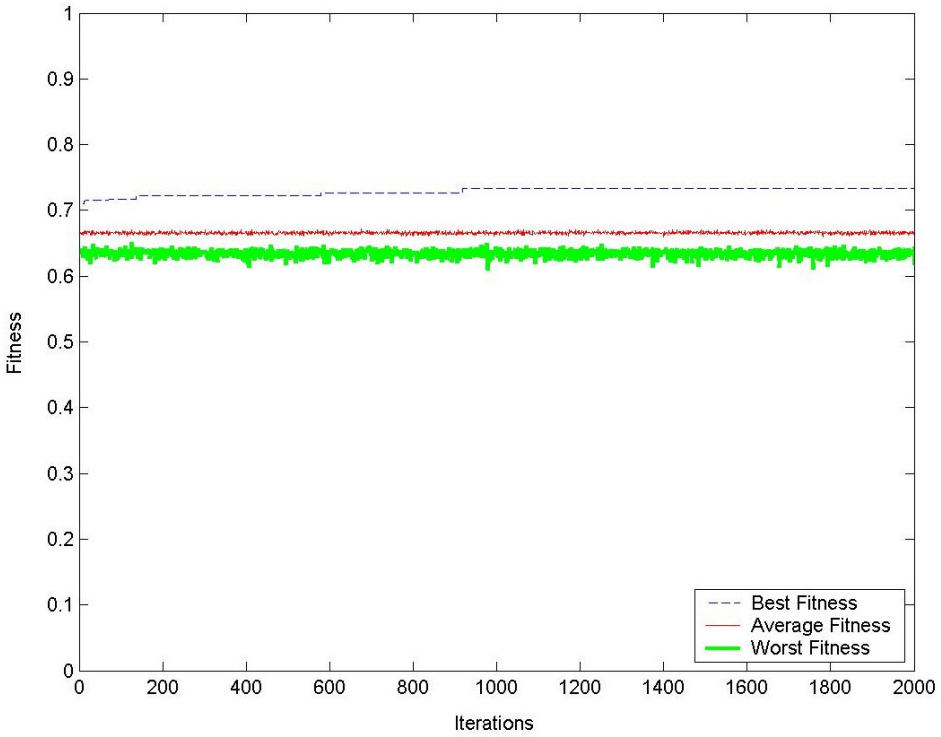
6.4 Extensions to the Initial System

6.4.1 Double Sided strips

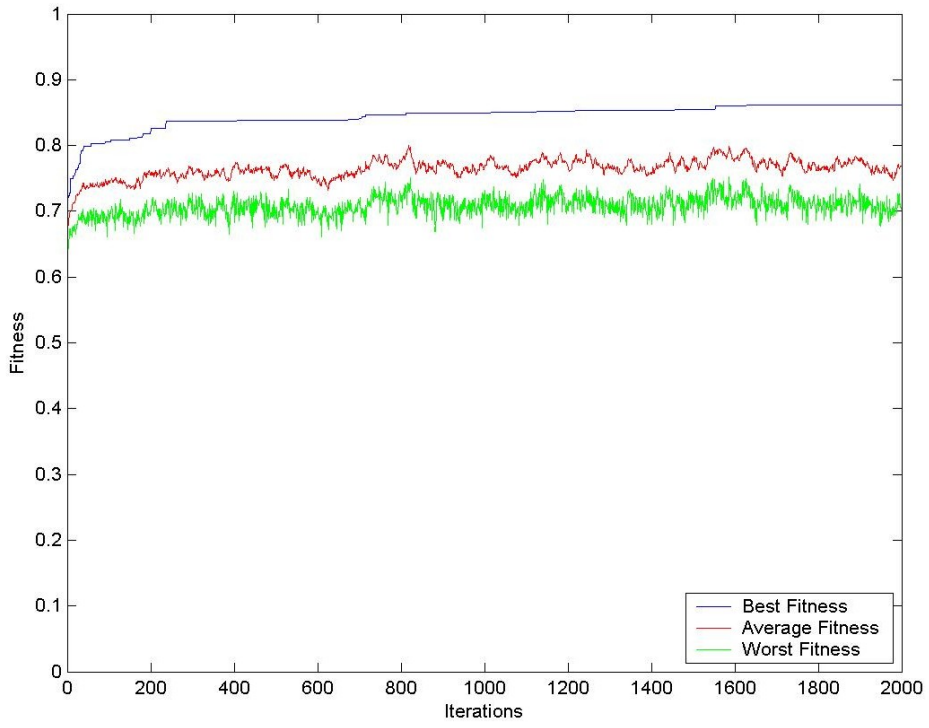
The first extension to the main software system was the ability to reconstruct double sided documents. Rather than scan in more documents, it was noted that double sided documents could be simulated, simply by using two existing documents which had the same number of strips. If the correct order of both of these documents was known, strips could be matched with each other as though they were double sided.

For example, given two documents:

$$d_1 = \begin{bmatrix} 1 & 3 & 5 & 2 & 6 & 4 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

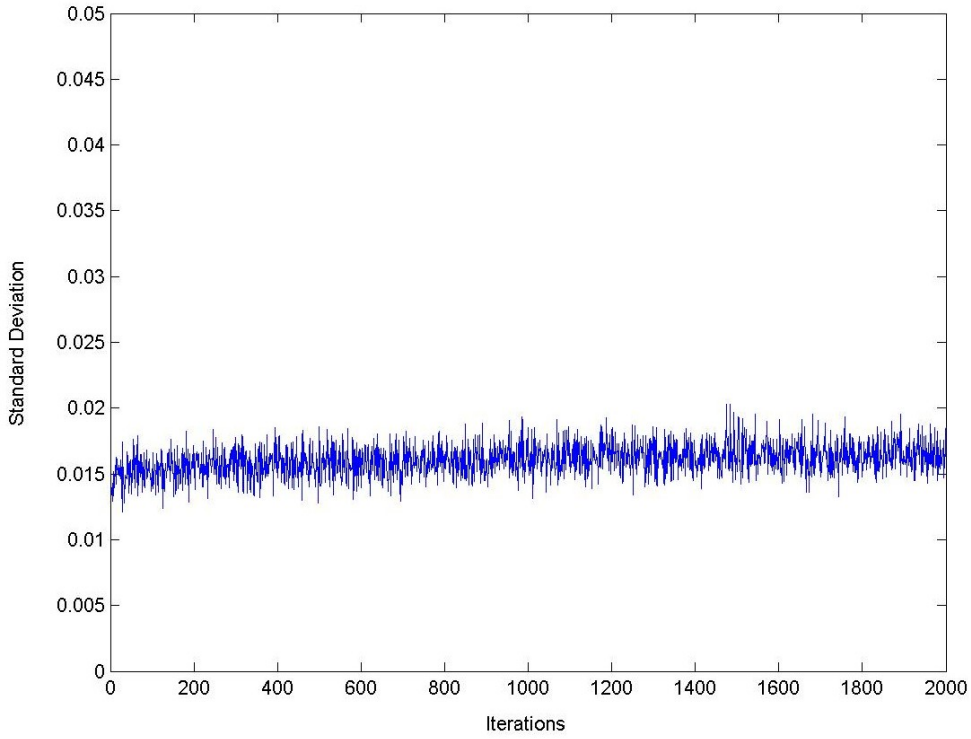


(a) Fitness of a random search: Shows little improvement over time

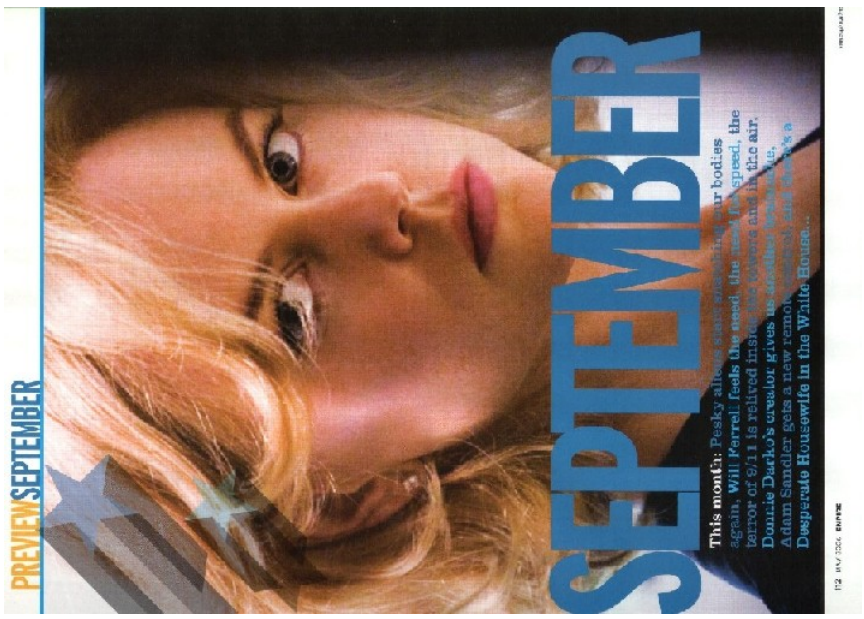


(b) Fitness of an intelligent search: Quick improvement initially, then levelling out over time

Figure 6.13: Comparison of a random search versus an intelligent search

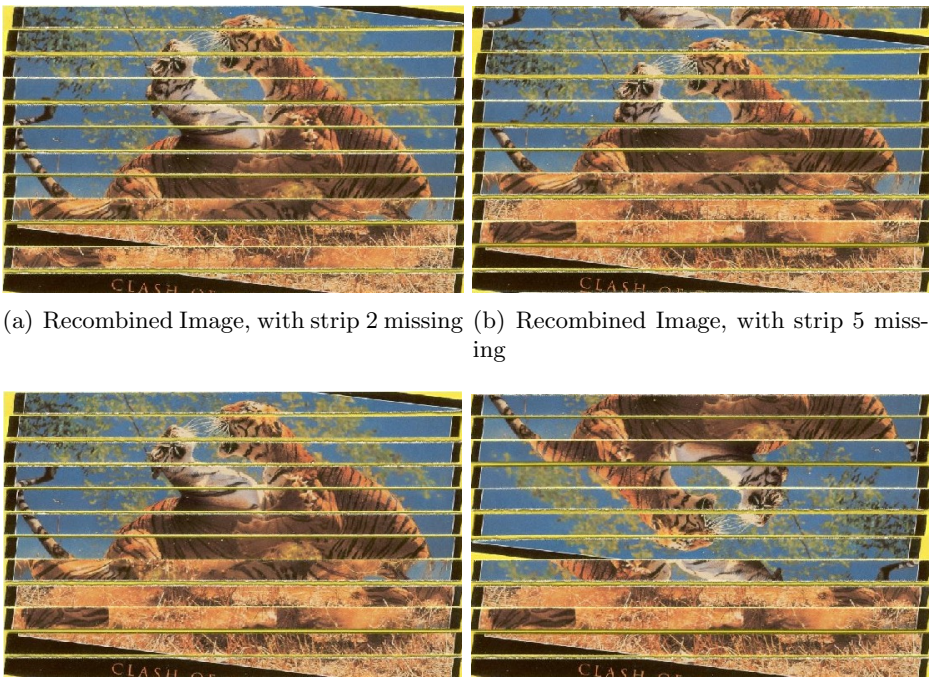


(a) Standard deviation



(b) Woman: the image being tested

Figure 6.14: Standard deviation of the fitness for the random search - shows little change over the run



(a) Recombined Image, with strip 2 missing (b) Recombined Image, with strip 5 missing

(c) Recombined Image, with strip 9 missing (d) Recombined Image, with strip 12 missing

Figure 6.15: Reconstruction with Missing Strips: Results are still fairly accurate, though (b) and (d) have reconstructed less well, most likely because the missing strips was more important to the reconstruction

$$d_2 = \begin{bmatrix} 3 & 2 & 6 & 1 & 4 & 5 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

If we consider d_1 as the reference frame, we can relate d_2 by noting the strips that match d_1 in the correct order: [3 1 2 5 6 4] — as piece 3 in d_2 matches piece 1 in d_1, piece 1 matches piece 2, etc. Rotation must also be taken into account, so if both strips have the same orientation, it would be marked as 0, and if the orientations are different, this is marked as 1:

$$\text{order} = \begin{bmatrix} 3 & 1 & 2 & 5 & 6 & 4 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

This ordering representation can then be used in the Genetic algorithm to create a population: a member of the population is created as normal, with strips and orientations chosen at random — these would represent d_1 — and the corresponding strips are created with the correct ordering relating to d_2:

$$\text{pop}_1 = \left[\begin{array}{cccccc} 5 & 2 & 1 & 3 & 6 & 4 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ \hline 6 & 1 & 3 & 2 & 4 & 5 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right]$$

This is a basic member of the population, and consequently correct fitness calculations can be made to determine the overall fitness of this combination of strips. The crossover methods were adapted as a result of these changes — the basic crossovers simply used the same methods as before, just working with one document, then used the ordering representation to determine the correct strips for the rest of the document.

The more intelligent crossover methods made use of the extra fitness information to increase the effectiveness of the search. For each edge, the fitness would be taken as the average of two individual fitnesses. This meant the potential for better searching, as correctly matched edges which had a poor fitness rating for one document might have a better fitness rating on the other. Therefore, it was considered likely that this would increase the accuracy and speed of the algorithm.

Testing of the Double-Sided System

All testing of the double-sided system was done using the Intelligent Edge Recombination Crossover. The main test was to see if combining the two images resulted in an increase in accuracy and/or speed. Both real and synthetic data was used.

Two synthetic images, Waterfall and Sunset (see Appendix A) were cut into 13 strips, and the GA was run on them, both individually, and as though they were a double sided document, and results were compared (Table 6.5). The same was done with two real images, Woman and Rainbow, and results were again compared (Table 6.6).

Image	Average Iterations required for Correct Solution	
	Individual	Double Sided
Waterfall	603	454
Sunset	1027	454

Table 6.5: Results of Double Sided Testing (Synthetic Data): The combined recombination converges on the correct solution more rapidly than the individual algorithms

While the speed of the system seems to increase when using synthetic data, performance is apparently less efficient on real data. However, when taking

Image	Highest Fitness Rating Found	
	Individual	Double Sided
Woman	0.8614	0.8607
Rainbow	0.8756	0.8696

Table 6.6: Results of Double Sided Testing: Fitness tends to be poorer with the combined method, however, looking at the results (Figure 6.16), they are similar in accuracy to the individual examples

into account the inaccuracies in the fitness evaluation, this result makes more sense — as both Woman and Rainbow have some poor comparisons which prevent the program finding a correct solution for them individually, these poor solutions are combined when comparing both together. Consequently, the fitnesses are lower.

However, when looking at the solutions for the double-sided test (Figure 6.16), the images are still fairly well reconstructed, and in the case of Rainbow, possibly a better solution that the one previously found, despite having a lower overall fitness. Most likely, with a more accurate fitness evaluation, performance would improve (as shown by the synthetic test results).

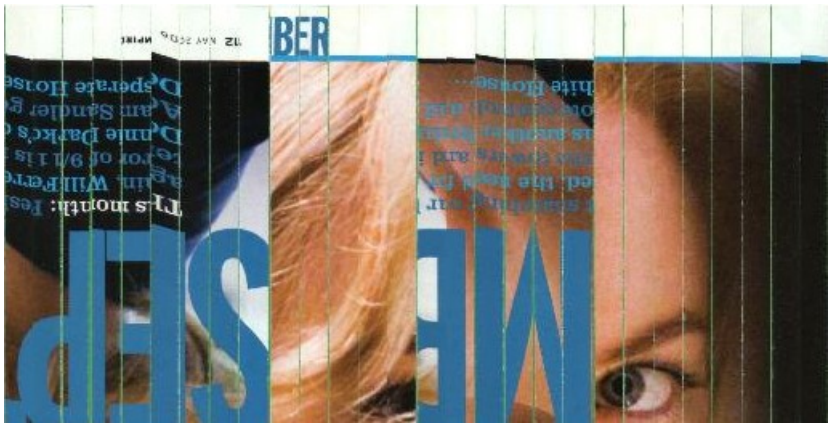
An interesting test to perform might be to combine real and synthetic data into a double sided image and analyse the results.

6.4.2 Multiple Documents

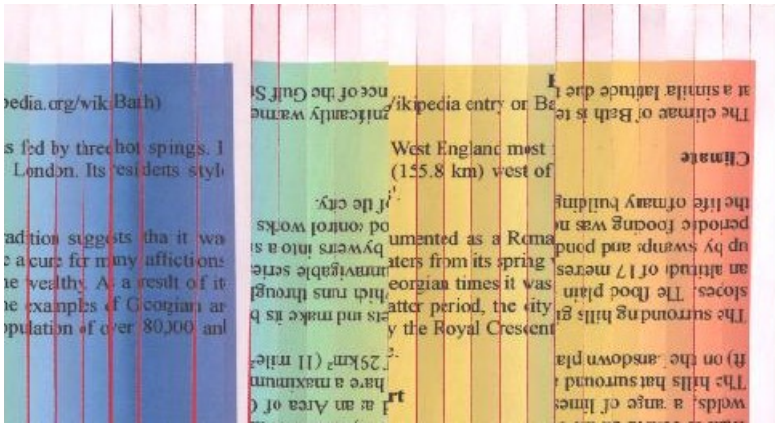
The final extension to the system was the capability to recombine multiple documents simultaneously. It was considered that any attempts to simply input all strips into the original program, and recreate the documents by representing them as one large document would be impractical, as the size of the problem space and computation required would almost certainly lead to poor results. Instead, it was concluded that strips could be split into groups, and solved simultaneously by GAs running in parallel. This would still require a large amount of computation, but the number of possible solutions would be reduced.

Therefore, having extracted the strips, they were separated into the correct number of groups (this depended on how many documents were expected in the output). This was done by computing colour histograms for each strip, and comparing the results. A strip was chosen at random, and all strips with histograms suitably similar were placed in the same group. Having created the groups, populations were created for each, and an iteration of the GA was performed for each group.

In order to compensate for inaccuracies in the splitting of the strips, the possibility of swapping a strip from one group to another was introduced.



(a) Part of Outputted Image - Woman



(b) Part of Outputted Image - Rainbow

Figure 6.16: Combined Reconstruction: While the overall fitness was lower, reconstruction was still fairly accurate, with both images split into 4 sections

Fitnesses were recorded for each strip at each iteration (only using selected members of the current population, as it was impractical to store all fitnesses for the entire population). Around every 20th iteration, these fitnesses were evaluated — strips which had consistently poor fitness ratings were considered to be incorrectly placed, and were consequently swapped with another strip in a different group.

Rather than calculating all fitness values in advance as before, calculations could also be performed while the GA was running — values would be stored in the correct position in the fitness table after being calculated, to prevent the need to calculate them twice. This was done to prevent some needless fitness calculations being done, as it was likely that some strips would remain in separate groups, and therefore never need to be compared.

Testing of the Multiple Document System

As before, testing of the multiple document system was done using the Intelligent Edge Recombination Crossover. The main test was of the effectiveness of the swapping function, both initially, and after a set number of iterations. Once this function was suitably accurate, testing could be performed on the full algorithm, if time allowed. Tests were again performed on both real and synthetic data.

Originally, tests were performed on synthetic images Tower and Purple, but the clear difference in colour meant that the split was simple for the colour histograms. Consequently, it was concluded that a better test of the system would include more similar images, to see how well the algorithm performed.

Two synthetic images, Tower and Waterfall were resized to make them of identical dimension, and split into 30 strips. They were separated into 2 groups initially, using colour histograms, and the results were recorded in Table 6.7.

Document	Strips	Incorrect Strips
1	1 4 5 6 8 9 10 11 12 13 14 16 17 18 20 21 22 23 24 25 26 27 28 29 33 36 42 43 53 60	6
2	2 3 7 15 19 30 31 32 34 35 37 38 39 40 41 44 45 46 47 48 49 50 51 52 54 55 56 57 58 59	6

Table 6.7: Splitting of 2 documents - Synthetic data: The split was fairly accurate, with only 6 strips placed into the wrong groups

The algorithm was run for 1,000 iterations 5 times, and on average, the number of strips incorrectly placed overall was 8 (4 in each document).

(a) First Image

(b) Second Image

Figure 6.17: Output of Multiple Document Program - Synthetic Data: Despite incorrect strips, the results were close to accurate

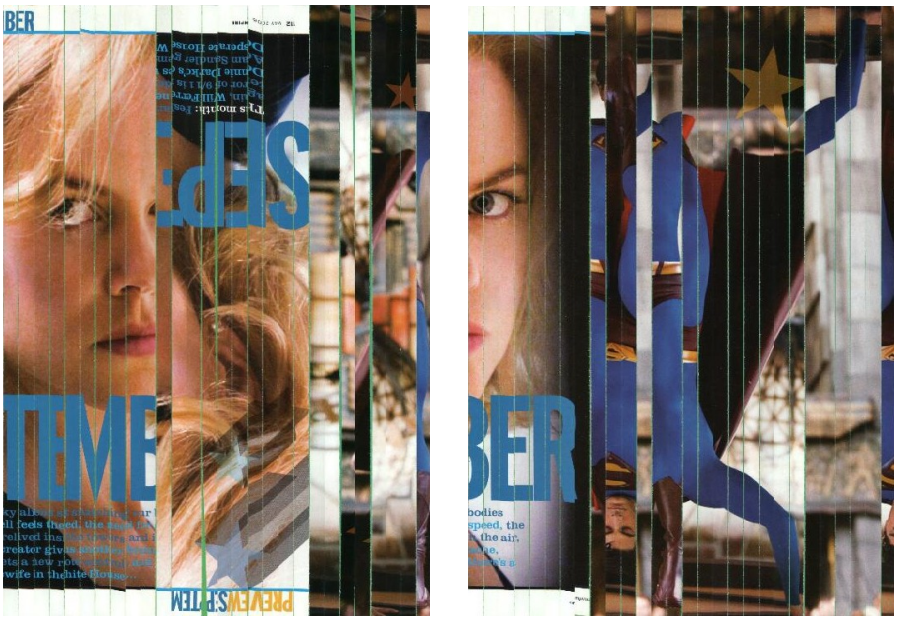
The solution with the fewest incorrectly placed strips was considered (see Figure 6.17).

Two real images, Woman and Superhero were split into 2 groups, and the results were recorded in Table 6.8.

Document	Strips	Incorrect Strips
1	1 2 5 6 7 8 9 10 11 12 14 16 17 18 19 22 23 24 26 28 29 31 33 39 44 48 53 54	8
2	3 4 13 15 20 21 25 27 30 32 34 35 36 37 38 40 41 42 43 45 46 47 49 50 51 52 55 56	8

Table 6.8: Splitting of 2 documents - Real data: The split was reasonably accurate, but showed little improvement over time

The algorithm was run for 1,000 iterations 5 times, and on average, the number of strips incorrectly placed was still 16 (8 in each document). The solution with the fewest incorrectly placed strips was considered (see Figure 6.18).



(a) First Image (b) Second Image

Figure 6.18: Output of Multiple Document Program - Real Data, 2 images: The number of inaccurate strips did not improve over time, though there are portions of both images correctly reconstructed

Three real images, Lines, Rainbow and Text were split into 3 groups (of

equal size, meaning that correct reconstruction would be impossible, as Lines contained only 26 strips, and Rainbow 28). The results of the initial split were recorded in Table 6.9.

Document	Strips	Incorrect Strips
1	1 5 6 9 10 11 12 13 14 15 16 17 18 20 23 24 26 31 32 35 38 39 40 42 46 48 49	10
2	2 4 7 8 27 28 29 30 33 34 36 37 41 43 44 45 47 50 51 52 53 54 55 58 59 68 70	10
3	3 19 21 22 25 56 57 60 61 62 63 64 65 66 67 69 71 72 73 74 75 76 77 78 79 80 81	5

Table 6.9: Splitting of 3 documents - Real data: The split was reasonably accurate, and showed some improvement over time

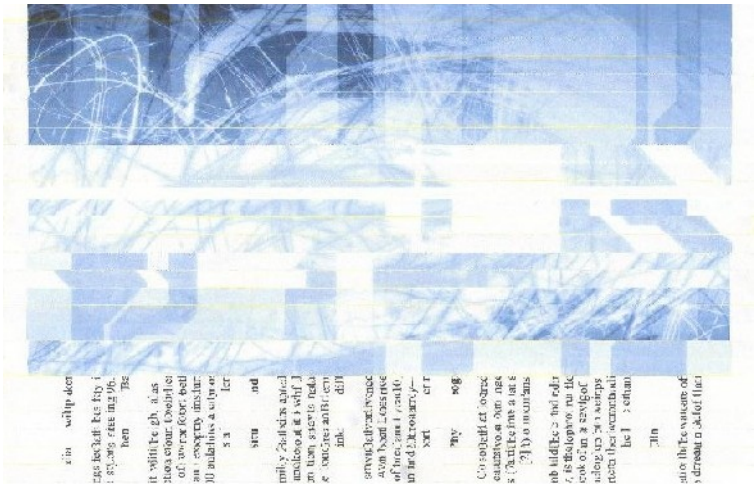
The algorithm was run for 1,000 iterations 5 times, and on average, the number of strips incorrectly placed was 20 — an improvement of 5 strips on the initial assignment. The solution with the fewest incorrectly placed strips was considered (see Figure 6.19).

Overall, the accuracy of the split was not too bad - generally, the majority of the strips were correctly placed. More experimentation will need to be performed to improve it sufficiently to provide correct solutions eventually. The original splitting is most likely the key to this, as a poor split initially will confuse the algorithm as to which are the correct groupings.

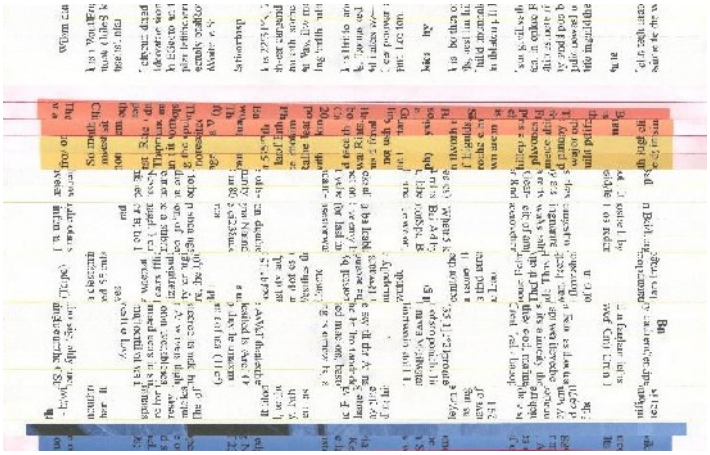
For example, if the strips are split as half from one document in one group, and half in another, which is the correct grouping? Most likely, there will be no way to converge on a correct solution, as there will be no consistently poor fitnesses to mark out a strip to be swapped.

It is also likely that this was the case with some of the testing examples, for example with Woman and Superhero. It can be seen in Figure 6.18(b) that the incorrect strips in the image have formed a correct portion of the other document. This is probably why the strips were not successfully swapped between the two documents.

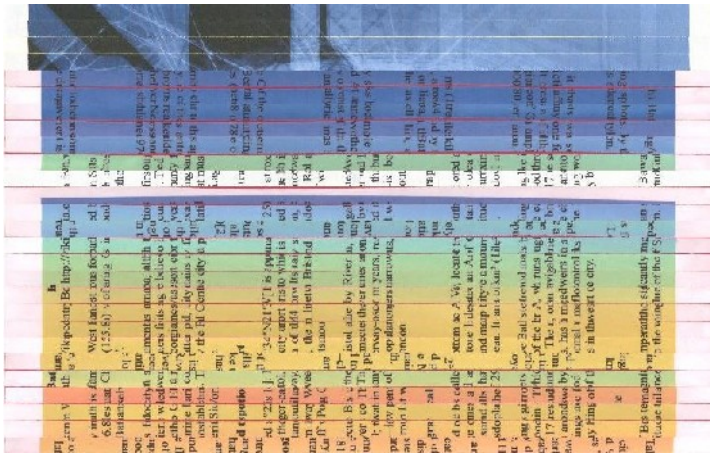
Possibly, the probability of swapping strips should be reduced as the algorithm runs — hopefully, by a certain point, all the strips will be correctly grouped, and swaps will not be necessary. Also, currently if the documents have different numbers of strips, there is no allowance made for changing the number of strips in the document. This is something to consider in the future.



(a) First Image



(b) Second Image



(c) Third Image

Figure 6.19: Output of Multiple Document Program - Real Data, 3 images: While overall reconstruction was poor, there are subsections correctly recombined. The text reconstruction is unimpressive, but the rainbow colours are reasonably matched

Chapter 7

Conclusions

7.1 Overall Conclusions on the System

The majority of the design specification was fulfilled — the system does take an image, process it, and return a completed image, which is usually at least partially correct. Extensions to the initial system were completed, and the system can deal with double-sided and multiple documents.

While the system has demonstrated some good results on several images, the ultimate conclusion is that it does not produce the correct results. Most likely, this is because of a poor fitness function, although it could also be partly because the extraction process can also be poor at times. The GA is fairly reliable — using the synthetic data, it does generally produce the correct solution (thought the number of iterations necessary can vary).

Despite this, some aspects of the system work well — it can clearly be seen that, when enough colour data is present, a good partial solution can be reached. Strip extraction performs well, with minimal background noise, and the intelligent GA can be relied upon to find a good solution in terms of fitness. However, performance on text-based images is generally poor. This is again due to the fitness function, as it is not equipped to deal with images with little colour or variation.

The system can be very slow to run in the extraction and fitness evaluation stages, particularly when dealing with A4-sized images. This is reasonable though, as A4 images include a large number of pixels, and usually contain around 26/27 strips. When comparing the performance of the system to reconstruction by hand however, there is really no competition. This is a problem, particularly when one of the aims of the project was to try and find a system comparable to reconstruction by hand.

7.2 Conclusions on Implementation and Project Planning

When considering the overall implementation of the project, it probably would have been better to have spent more time developing the evaluation function — that was the major weakness in the final system. Rather than performing an in-depth study into the main problem, the emphasis became on extending the project to include all of the desired outcomes. As a result, some areas were not examined as fully as they could have been, and the final results suffered as a consequence.

Looking at the original project plan (see Appendix B and Section 3.7), the schedule was kept to fairly well. Some areas did overrun slightly, and the time available during the exam period was overestimated, but generally, there were no major setbacks.

7.3 Improvements to the System

The most pressing area for improvement would be the fitness function. If it could be extended to deal with text-based images, and provide more accurate evaluation of images generally, the system would be vastly improved. Possibly, content-based image retrieval could be considered — indeed, this has been considered in previous studies into this problem. Character or text recognition might be another consideration, although this would limit the system in terms of the range of images it could deal with — it would only work for text it had been trained on.

The strip extraction could be made more robust, as the current method of corner detection is not completely reliable. Also, as mentioned before, it might be beneficial to allow irregularly shaped strips into the system. These could possibly be extracted using active contours, or a similar method.

Currently, the only termination clause for the program is either reaching a specified number of iterations, or in the case of synthetic data, reaching the maximum possible fitness. This could be improved by possibly stopping the iterations after diversity has reached a certain level, or if average fitness has not improved over time.

The GA could also be improved by reducing the time needed to find the correct solution. Possibly, the Intelligent Ordered Crossover could be made more efficient, as it is significantly faster than the Edge Recombination, despite having poorer performance. There are other potential changes — while the Roulette Wheel selection works sufficiently well, there are other methods that could be considered (Tournament selection, or Stochastic Universal selection).

Rather than using random selection to choose the initial population, it could be done using more intelligent methods, meaning that the system begins with some good parents to breed with. More adaptive mutation could be done, depending on the current diversity of the population. All of these could potentially improve overall performance.

Another way to improve the speed of the system is to consider changing the language to something more efficient, such as C++ or C, or to use distributed systems to run the algorithm. The multiple document program would be well suited to this, as the program currently uses a parallel GA to find solutions.

7.4 General Conclusions on the Design

While the system that was implemented did work reasonably effectively, when considering the general problem of document reconstruction, it is possible that this method is not the most useful. Given only 20 or 30 strips, even only containing text, most people can reconstruct a document in a short space of time. While using more strips makes the task more difficult, any program which tries to reconstruct these strips would still probably be considerably slower.

The possibility of dealing with thousands of strips, which is possible when searching through for example, a company's shredded documents, makes the method used in this project impractical. Therefore, a more useful area to consider in terms of the real-world problem could be that of separating multiple documents into correct groups. Given the correct strips, a person could then complete the process quickly by hand. Some investigation was conducted into this during the multiple document reconstruction, though it was mainly for the purposes of then using the grouped strips to reconstruct a final document.

While the separation was occasionally good (particularly when dealing with synthetic images, as would be expected), this would certainly be an area of improvement for the project in the future. It would probably be more practical to look into document separation rather than reconstruction when considering much larger numbers of strips, should the project be continued. There are some examples of these methods being used more extensively to solve this problem in the literature, mentioned previously [4].

7.5 Future Work

Possible areas to examine in the future might include further consideration of the effects of missing strips on reconstruction. While testing showed there

was not too much effect on the results with only one piece missing, but could there be a minimum number of strips necessary to construct a reasonable result? What about if there is a strip that does not belong in the document — will this skew the results?

There is the possibility of dealing with double-sided documents when it is unclear which side a piece belongs on. The current method assumes that all sides are correctly sorted, and belong to the correct document. If this was not the case, the possibility of flipping a strip over would need to be considered. To implement, this would probably require a hybrid of the double-sided method, and the multiple document method.

Another extension to consider could be dealing with cross-cut shredded documents. This would obviously add complexity to the problem, but would the additional fitness evaluations help to improve results? For example, with strip-shredded documents, the are only one or two comparisons that can be used to judge if a strip is in the correct place. With crosscut shredding, there are potentially four adjacent pieces to compare to — would this make reconstruction easier?

Bibliography

[1] De Smet, P., De Bock, J., Quang Luong, H., 2005, On the reconstruction of strip-shredded documents, *1st annual IEEE BENELUX/DSP Valley Signal Processing Symposium (SPS-DARTS 2005)*, April 19-20, 2005, Belgium. (Awaiting publication)

[2] De Smet, P., 2005, Automated reconstruction of strip-shredded documents, *American Academy of Forensic Sciences, 57th Annual Meeting (AAFS2005)*,21-26 February 2005, New Orleans, USA. (Awaiting publication)

[3] Justino, E., Oliveira, L.S., Frei, C., 2005, Reconstructing shredded documents through feature matching, *Forensic Science International, In Press* Available from: http://www.sciencedirect.com/science/article/B6T6W-4HDG9H9-1/2/70d5747569ae5f9bd1840ba4f301c0d9 [Accessed 13 Nov 2005]

[4] Ukovich, A., Ramponi, G., Doulaverakis, H., Kompatsiaris, Y., Strintzis, M.G., 2004, Shredded document reconstruction using MPEG-7 standard descriptors, *IEEE Int. Symp. on Signal Processing and Information Technology — ISSPIT—04*, 18-21 Dec. 2004, Rome, Italy.

[5] Freeman, H., Garder, L., 1964, Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition, *IEEE Transactions on Electronic Computers*, vol. 13, pp. 11827, April 1964.

[6] Goldberg, D.E., Malon, C., Bern, N., 2002. A Global Approach to the Automated Solution of Jigsaw Puzzles. *Symposium on Computational Geometry*, 5-7 June 2002, Barcelona, Spain.

[7] Kosiba, D., Devaux, P., Balasubramanian, S., Gandhi, T., Kasturi, R., 1994, An automatic jigsaw puzzle solver, *Proceedings of the 12th IAPR International Conference on Pattern Recognition, 1994. Conference A: Computer Vision and Image Processing*, vol. 1, pp. 61618.

[8] Chung, M.G., Fleck, M.M., Forsyth D.A, 1998, Jigsaw Puzzle Solver Using Shape and Color, *Fourth International Conference on Signal Processing Proceedings, (ICSP '98)*, vol. 2, pp. 877-80, 12-16 Oct. 1998.

[9] Toyama, F., Fujiki, Y., Shoji, K., Miyamichi, J., 2002, Assembly of puzzles using a genetic algorithm. *16th International Conference on Pattern Recognition, 2002. Proceedings.* vol. 1, pp. 389-92.

[10] Nixon, M., Aguado, A., 2002. *Feature Extraction and Image Processing.* Oxford: Newnes. ISBN 0-7506-5078-8.

[11] Seul, M., O'Gorman, L., Sammon, M.J., 2000. *Practical algorithms for image analysis : description, examples, and code.* Cambridge: Cambridge University Press. ISBN 0-5216-6065-3.

[12] Russ, J.C., 1995. *The image processing handbook.* London: CRC Press. ISBN 0-8493-2516-1.

[13] Freeman, H., 1961, On the Encoding of Arbitrary Geometric Configurations *IRE Transactions on Electrical Computers*, vol 10, pp 260-68, April 1964.

[14] Roberts, L.G., 1965, Machine Perception of Three-Dimensional Solids, *Optical and Electro-Optical Information Processing*, MIT Press, pp 159-197.

[15] Sobel, I.E., 1970, *Camera Models and Machine Perception*, PhD Thesis, Stanford University.

[16] Canny, J., 1986, A Computational Approach to Edge Detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 8, No. 6, pp. 679-98.

[17] Harris, C., Stephens, M., 1988, A Combined Corner and Edge Detector, *Proceedings of the 4th Alvey Vision Conference*, pp. 147-51.

[18] Hough, P.V.C., 1959, Machine Analysis of Bubble Chamber Pictures, *International Conference on High Energy Accelerators and Instrumentation*, CERN, 1959.

[19] Ballard, D., 1981, Generalized hough transform to detect arbitrary patterns, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13(2), pp. 11122.

[20] Dehong, L., Lihan, H., Carin, L., 2004, Airport detection in large aerial optical imagery *IEEE International Conference on Acoustics, Speech, and Signal Processing, Proceedings. (ICASSP '04)*, vol.5, pp. 761-4.

[21] Kass, M., Witkin, A., Terzopoulos, D., 1987, Snakes: Active contour models, *Proceedings of the First International Conference on Computer Vision*, Vol. 1, pp. 259-68.

[22] Garey, M.R., Johnson, D.S., 1979, *Computers and Intractability A Guide to the Theory of NP-Completeness*. Freeman. ISBN 0-7167-1044-7.

[23] Blum, C., Roli, A., 2003, Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3), pp. 268308.

[24] Mitchell, M., Holland, J.H., Forrest, S., 1994, When will a genetic algorithm outperform hill climbing?, *In*: Cowan, J.D., Tesauro, G., Alspector, J., (eds.) *Advances in Neural Information Processing Systems*, Vol. 6. San Mateo, CA: Morgan Kaufmann.

[25] Talbi, E.-G., Muntean, T., 1993, Hill-climbing, simulated annealing and genetic algorithms: a comparative study and application to the mapping problem, *Proceeding of the Twenty-Sixth Hawaii International Conference on System Sciences*, Volume ii, 5-8 Jan. 1993, pp.565-73.

[26] Ackley, D.H., 1987, An Empirical Study of Bit Vector Function Optimization, *In*: Davis, L., (ed.), *Genetic Algorithms and Simulated Annealing*. London: Pitman. ISBN 0-2730-8771-1.

[27] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983, Optimization by Simulated Annealing. *Science*, vol. 220, pp. 671-80.

[28] Cerny, V., Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, vol. 45, pp. 41-51.

[29] Glover, F., Laguna, M., 1997. *Tabu Search*. 2nd Edition. Boston : Kluwer Academic Publishers. ISBN 0-7923-9965-X.

[30] Glover, F., Rego, C., 2004, Illustrative Tabu Search Applications http://www.tabusearch.net/tabu_search/applications.asp [Accessed 15 Nov 2005]

[31] Holland, J.H., 1975, *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press. ISBN 0-2625-8111-6.

[32] Davis, L., 1991, *Handbook of genetic algorithms*. New York : Van Nostrand Reinhold. ISBN 0-4420-0173-8.

[33] Reeves, C.R., Rowe, J.E., 2003, *Genetic algorithms : principles and perspectives : a guide to GA theory*. London : Kluwer Academic. ISBN 1-4020-7240-6.

[34] Dorigo, M., Maniezzo, V., Colorni, A., 1991, Positive Feedback as a Search Strategy, *Technical Report No. 91-016*, Politecnico di Milano, Italy. Available from: http://www.aco-metaheuristic.org/publications.html [Accessed on 15 Nov 2005]

[35] Hart, W.E., 1996, A Theoretical Comparison of Evolutionary Algorithms and Simulated Annealing, *In*: Fogel, L.J., Angeline, P.J., Back, T., (eds.), *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, San Diego, CA, USA, February 29 - March 2, 1996. MIT Press.

[36] Larranaga, P., Kuijpers, C. M. H.,Murga, R. H., Inza, I.,Dizdarevic, S., 1999, Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators, *Artificial Intelligence Review*. 13, pp. 129-170.

[37] Davis, L., 1985, Applying Adaptive Algorithms to Epistatic Domains, *Proceedings of the International Joint Conference on Artificial Intelligence* pp. 162-164

[38] Sonka, M., Hlavac, V., Boyle, R., 1999, *Image Processing, Analysis, and Machine Vision*. 2nd Edition. California : Brooks/Cole. ISBN 0-534-95393-X.

[39] Mancas-Thillou, C., Gosselin, B., 2005, Color Text Extraction from Camera-based Images the Impact of the Choice of the Clustering Distance, *Proceedings of the Eighth International Conference on Document Analysis and Recognition*. pp. 312-316

[40] Goldberg, D.E., Deb, K., Clark, J.H., 1992, Genetic algorithms, noise, and the sizing of populations, *Complex Systems*, vol 6, pp. 333-362.

Appendix A

Testing Images

This is a list of the images used for testing purposes in the project. Images A.1-A.5 are synthetic, and images A.6-A.15 are real.

Figure A.1: Tower: Chosen as a good general image, with a good range of colour and texture.

Figure A.2: Waterfall: Chosen because of the good range of textures, but slightly more limited colours

Figure A.3: New York: Chosen as a good general image, with a good range of colour and texture.

Figure A.4: Purple: Chosen because of the limited range of colour and texture

Figure A.5: Sunset: Chosen because of the limited range of colour and texture

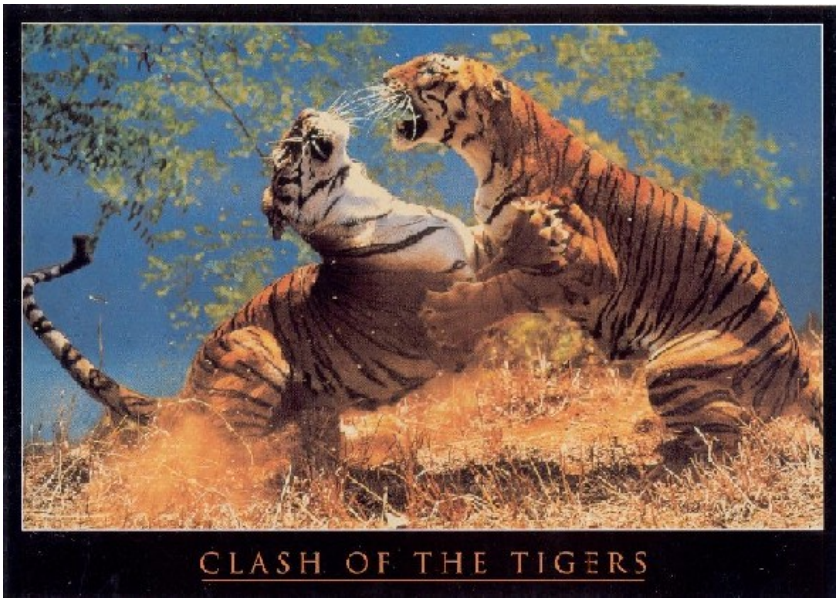


Figure A.6: Tigers: Split into 13 strips. Chosen for the strong colours, and also as the strip were shredded at a skewed angle, making them not entirely rectangular

Figure A.7: Planet: Split into 12 strips. Chosen for the combination of text and image.

Figure A.8: Ape: Split into 14 strips. Chosen for the limited range of colours, but with a strong image in the foreground.

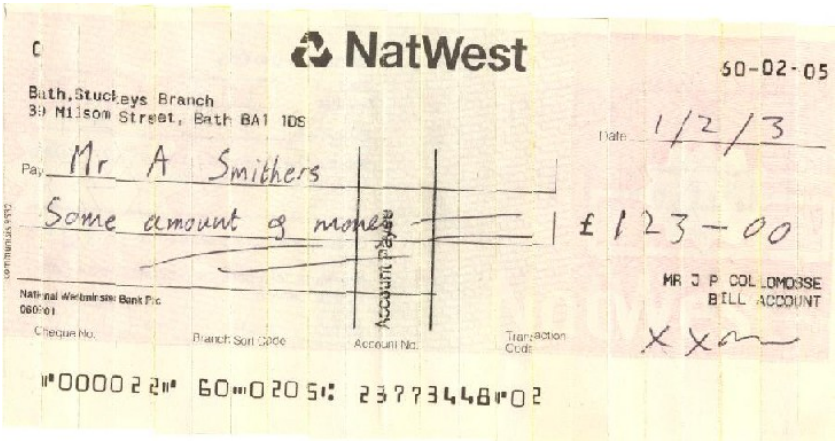
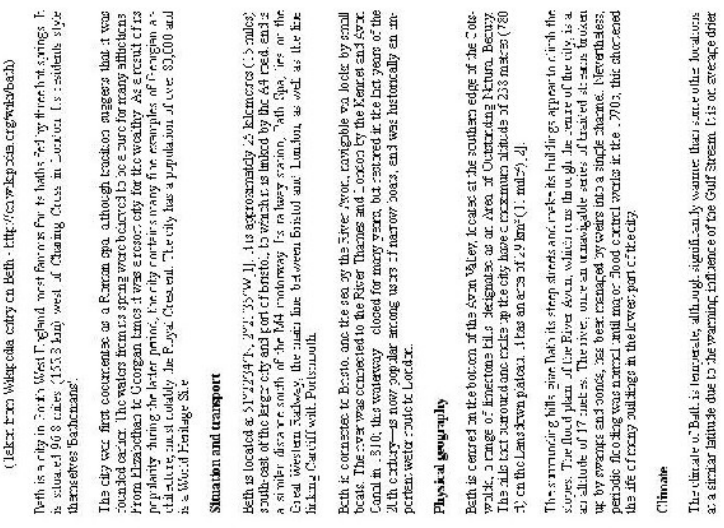


Figure A.9: Cheque (original image unavailable): Split in 22 strips. Chosen for limited range of colour, and handwritten text.

Figure A.10: Girl: Split into 16 strips. Chosen as a strong image to reconstruct.



Bath

(Redirected from Wikipedia:Citing Bath - http://en.wikipedia.org/wiki/Bath)

Bath is a city in South West England most famous for its baths fed by three hot springs. It is situated 96 miles (153.3 km) west of Charing Cross in London. Its residents style themselves 'Bathonians'.

The city was first occupied as a Roman spa, although tradition suggests that it was founded earlier. The waters from its spring were believed to be a cure for many ailments. From Elizabethan to Georgian times it was a resort only for the wealthy. As a result of its popularity during the latter period, literally hundreds of examples of Georgian architecture, most notably the Royal Crescent. The city has a population of over 83,000 and is a World Heritage Site.

Situation and transport

Bath is located at 51°22'54"N, 2°21'35"W [1]. It is approximately 25 kilometres (15 miles) south-east of the larger city and port of Bristol, to which it is linked by the A4 road and a similar distance south of the M4 motorway. Its railway station, Bath Spa, lies on the Great Western Railway; the main line between Bristol and London, as well as the line linking Cardiff with Portsmouth.

Bath is connected to Bristol and the sea by the River Avon, navigable via locks by small boats. The river was connected to the River Thames and London by the Kennet and Avon Canal in 1810, the waterway closed for many years, but restored in the last years of the 20th century—is now popular among users of narrow boats, and was historically an important water route to London.

Physical geography

Bath is centred on the bottom of the Avon Valley, located at the southern edge of the Cotswolds, a range of limestone hills designated as an Area of Outstanding Natural Beauty. The old city surrounds the spa on the city have a maximum altitude of 238 metres (780 ft) on the Lansdown plateau. It has an area of 29 km² (11 mi²). [1]

The surrounding hills give Bath its steep streets and crescents; its buildings appear to climb the slopes. The flood plain of the River Avon, which runs through the centre of the city, is at an altitude of 17 metres. The river, once an unnavigable series of tidal streams, was tamed by barrages and locks, and is best managed by weirs and a single sluice. Nevertheless, periodic flooding was normal until major flood control works in the 1970s, this shortened the life of many buildings in the lower part of the city.

Climate

The climate of Bath is temperate, although significantly warmer than most other locations at a similar latitude due to the warming influence of the Gulf Stream. It is on average drier

Figure A.11: Text: Split into 27 strips. Chosen as an example of a text-only image.

Figure A.12: Superhero: Split into 28 strips. Chosen for the strong foreground image, but limited background detail.

Figure A.13: Lines: Split into 26 strips. Chosen for the limited range of colour, but good pattern.

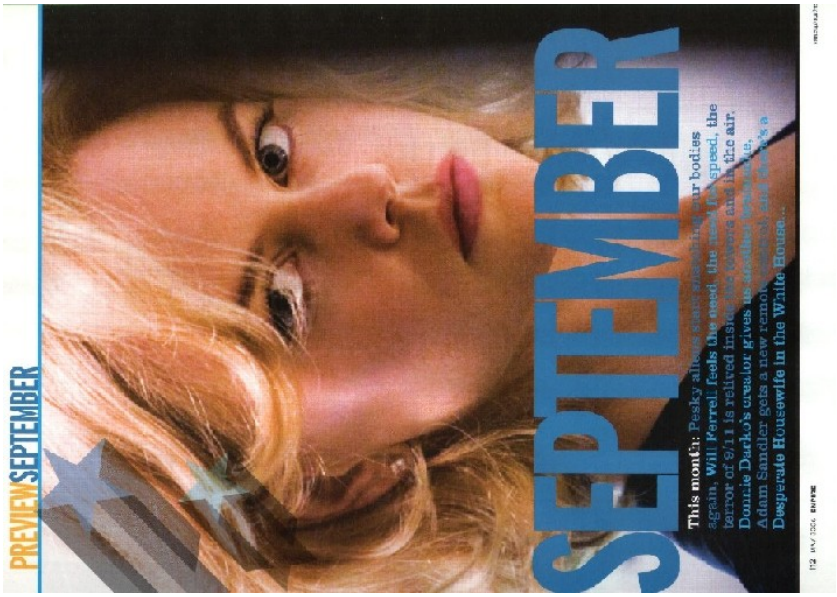


Figure A.14: Woman: Split into 28 strips. Chosen as a strong image to reconstruct, and as an example of text on an image.

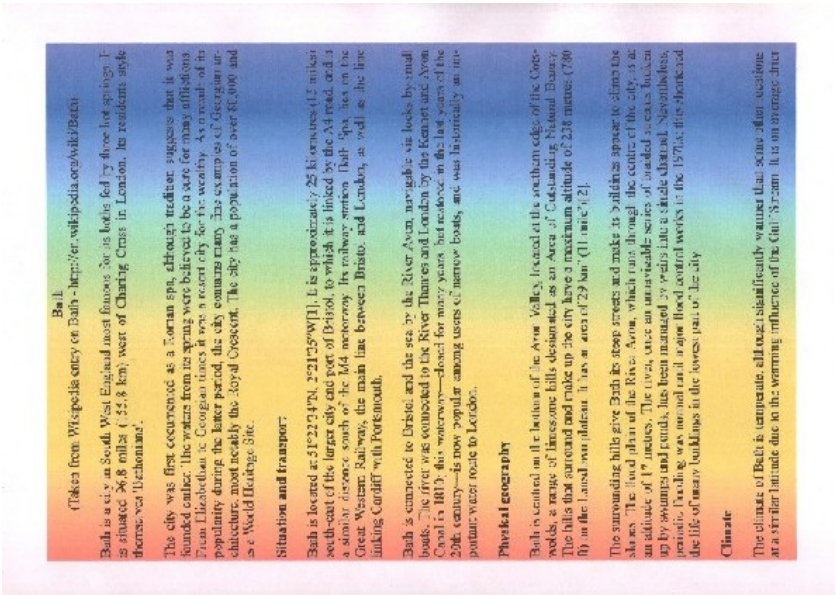


Figure A.15: Rainbow: Split into 28 strips. Chosen as an example of text, with a strong background.

Appendix B

Project Plan

The next page is the Gantt chart showing the original time plan for the project.

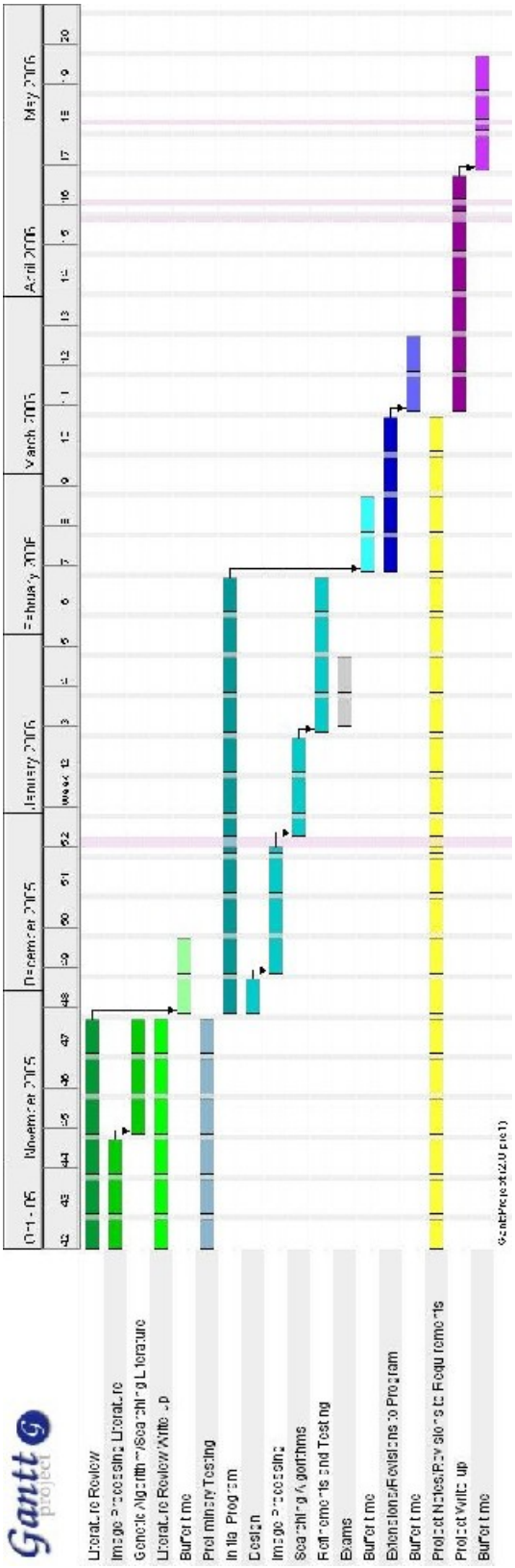

Gantt project
November 2005
December 2005
January 2006
February 2006
March 2006
April 2006
May 2006
Literature Review
Image Processing Literature
Genetic Algorithm Searching Literature
Literature Review Write up
Buffer time
Preliminary Testing
Initial Program
Design
Image Processing
Searching Algorithms
Refinements and Testing
Exams
Buffer time
Extensions/Revisions to Program
Buffer time
Project Notes/Presentation Requirements
Project Write up
Buffer time

Figure B.1:

Appendix C

Program Code

Program Code is too extensive to be included in this Appendix — all the final code for the project, and the testing images, are found on the attached CD.